[illegible]

[illegible] 1[illegible] [illegible]

[illegible]

[illegible]

[illegible] OOP[illegible] OOP[illegible]

[illegible] 2[illegible]

1.1 [illegible]

C++[illegible]

1.1.1 [illegible]+[illegible][1]

[illegible] 60[illegible] Simula-67[illegible] “[illegible]”[illegible] “[illegible]”[illegible] “[illegible]” [illegible]

[illegible]

[illegible] OOP[illegible] OOP[illegible] Smalltalk——C++[illegible] OOP[illegible]

[illegible] “[illegible]” [illegible] “[illegible]”[illegible] “[illegible]” [illegible] “[illegible]”[illegible]

1.1.2 [illegible]

[illegible]

---

[1] [illegible] The Tao of Objects[illegible] Gary Entsminger[illegible]

[illegible]

[illegible]“[illegible]”[illegible]

[illegible]“[illegible]”[illegible]

[illegible]

1.1.3 [illegible]

[illegible]

[illegible]“[illegible]”[illegible]

[illegible]

[illegible]
OOP[illegible]“[illegible]”[illegible] OOP[illegible]

[illegible]“[illegible]”[illegible]

[illegible]

[illegible] 14 [illegible]

[illegible] virtual [illegible] virtual [illegible] C++ [illegible] Virtual [illegible]

1.1.4 [illegible] OOP [illegible]

[illegible] C [illegible]

[illegible] C++ [illegible] C [illegible] C++ [illegible] main() [illegible] C [illegible] C++ [illegible] C [illegible]

1.2 [illegible] C++ [illegible]

C++ [illegible] C [illegible] OOP [illegible] C [illegible] OOP [illegible] C [illegible] C [illegible] C [illegible] C++ [illegible] OOP [illegible] C++ [illegible]

C++ [illegible] C++ [illegible]

1.2.1 [illegible] C

[illegible] C++ [illegible] C [illegible] C++ [illegible] C [illegible]

[illegible] C++[illegible] references([illegible])[illegible] C[illegible]

1.2.2 [illegible]

[illegible] C++[illegible] C[illegible] C++[illegible]

[illegible] C[illegible] C++[illegible] C++[illegible]

1.2.3 [illegible]

[illegible] C++[illegible] C[illegible] C++[illegible]

C++[illegible] C[illegible] C++[illegible] C++[illegible] C[illegible] ± 10%[illegible] OOP[illegible] C[illegible]

1.2.4 [illegible]

[illegible] “[illegible]”[illegible] “[illegible]”[illegible]

[illegible]

1.2.5 “[illegible]”[illegible]

[illegible] C++[illegible]

[illegible] C[illegible]

- [illegible]

[illegible]

[illegible]

1.2.6 [illegible]

[illegible] C [illegible] C++ [illegible] 17 [illegible]

1.2.7 [illegible]

[illegible] BASIC [illegible] C [illegible] 50 000 [illegible] C [illegible] —— [illegible]

[illegible] “[illegible] BASIC [illegible] C [illegible]” [illegible]

[illegible] C++ [illegible] hello-world [illegible] OOP [illegible]

1.3 [illegible]

[illegible] OOP [illegible] C++ [illegible] C++ [illegible]

[illegible] “[illegible]” [illegible] C++ [illegible]

1.3.1 [illegible]

[illegible]

[illegible]

1) [illegible]

2) [illegible] “[illegible]” [illegible]

[illegible]

1.3.2 [illegible]

[illegible] C++[illegible]

[illegible] 50[illegible] FORTRAN[illegible] FORmula-TRANslation[illegible] COBOL[illegible] COmmon Business-Oriented Language[illegible] Lisp[illegible] List-Processing[illegible] APL[illegible] A Programming Language[illegible]

[illegible] Lisp[illegible] APL[illegible] ——[illegible] “ mission[illegible] ”[illegible] FORTRAN[illegible] COBOL[illegible] FORTRAN[illegible] COBOL[illegible]

[illegible] Algol[illegible] Pascal[illegible] Prolog[illegible] FORTH[illegible] FORTH[illegible]

[illegible] BASIC[illegible] Beginners All-purpose Symbolic Instruction Code[illegible] 60[illegible] APL[illegible] “[illegible] ”[illegible] BASIC[illegible] APL[illegible]

[illegible]

[illegible] “[illegible] ”[illegible]

[illegible]

[illegible]

1.3.3 [illegible]

[illegible]

[illegible]

[illegible] FoobleBlah [illegible]

[illegible]

[illegible]

[illegible]

1. [illegible]

[illegible]

[illegible]

[illegible]

1) [illegible]

2) [illegible]

2. [illegible]

[illegible]“[illegible]”[illegible]“[illegible]”[illegible]

[illegible]“[illegible]”[illegible]80%[illegible]20%[illegible]20%[illegible]

3. OOP[illegible]

[illegible]——[illegible]

[illegible]

[illegible]OOP[illegible]“[illegible]”[illegible]OOP[illegible]

[illegible]

[illegible] 6[illegible] 50[illegible] 50[illegible]

[illegible] OOP[illegible] OOP[illegible] OOP[illegible]

1.3.4 [illegible]

[illegible] [1][illegible]

[illegible]

1) [illegible]

2) [illegible]

3) [illegible]

4) [illegible]

5) [illegible]

[illegible]

[illegible]

1) [illegible]

2) [illegible]

3) [illegible]

4) [illegible] ——[illegible]

5) [illegible]

---

[1] [illegible] Design Patterns:Elements of Reusable Object-Oriented Software by Erich Gamma et al., Addison-Wesley, 1995[illegible]

[illegible]

1.3.5 [illegible]

[illegible]

1. [illegible]

[illegible]“[illegible]”[illegible] OOP[illegible] C++[illegible] [1][illegible]

2. [illegible]

[illegible] OOP[illegible]

1.3.6 [illegible]

[illegible] ——[illegible]“[illegible]”[illegible]

1. [illegible]

[illegible] C++[illegible]

[illegible]“[illegible]”[illegible]

1) [illegible]

2) [illegible]“[illegible]”[illegible]

---

[1] A reference to vampires made in The Mythical Man-Month, by Fred Brooks, Addision-Wesley, 1975[illegible]

[illegible]

2. [illegible]

[illegible]

1) [illegible]

2) [illegible]

[illegible]

3. [illegible]

[illegible]

[illegible]

1) [illegible] “[illegible]” [illegible]

2) [illegible]

3) [illegible] “[illegible]” [illegible]

4) [illegible]

5) [illegible] C++ [illegible] OOP [illegible]

6) [illegible]

7) [illegible]

4. [illegible]

[illegible] Michael Wilk [illegible]

[illegible] “[illegible]” [illegible] “[illegible]” [illegible]

[illegible] OOP [illegible]

5. “[illegible]” [illegible]

[illegible]

[illegible]“[illegible]”[illegible]“[illegible]”[illegible]

[illegible]Software Creativity[illegible]Robert Glass[illegible]Prentice-Hall,1995[illegible]Glass[illegible]P.J.Plauger[illegible][1][illegible]

[illegible]Peopleware[illegible]Tom Demarco[illegible]Timothy Lister[illegible]Dorset House,1987[illegible]XYZ[illegible]

[illegible]Complexity, M. Mitchell Waldrop[illegible]Simon & Schuster, 1992[illegible]Santa Fe, New Mexico[illegible]

1.4 [illegible]

[illegible]——[illegible]“[illegible]”[illegible]“[illegible]”[illegible]Robert McKee[2][illegible]

[illegible]McKee[illegible]

[illegible]

---

[1] [illegible]“[illegible]”[illegible]Object Lessons Tom Love[illegible], SIGS Books, 1993[illegible]

[2] Through Two Arts, Inc.,12021 Wilshire Blvd. Suite 868, Los Angeles, CA 90025[illegible]

1.4.1 [illegible]

[illegible]

1) [illegible] C++[illegible]

2) [illegible] C++[illegible] “[illegible]” [illegible]

[illegible] DBMS[illegible] DBMS[illegible] wild-card[illegible] wild-card[illegible] —— [illegible][1][illegible] wild-card[illegible] “[illegible]” [illegible] wild-card[illegible]

[illegible] wild-card[illegible] “[illegible]”[illegible] OOP[illegible]

[illegible]

[illegible]

[illegible] Brooks[2] [illegible]“[illegible] —— [illegible] ……[illegible]”

[illegible]

[illegible] OOP[illegible]

---

[1] [illegible] wild-card[illegible]

[2] The Mythical Man-Month, [illegible]

□□□□□□□OOP□□[1]□□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□[2]□□

1.4.2 □□□

□□□

□□“□□□□□□□□”□□□“□□□□□□□□□□□□□□□□□□□”

1.4.3 □□(treatment)

□□□

□□□

1.4.4 □□□

□□

1. □□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

---

[1] □□□□□□□□□□□□□Microsoft Word□□□□□□□□□□□□□□□□□□□□□□□□□□□

[2] □□

[illegible]“[illegible]”[illegible]“[illegible]”[illegible]“[illegible]”[illegible]“[illegible]”[illegible]“[illegible]”[illegible]

[illegible]

2. [illegible]

[illegible]

[illegible]“[illegible]”[illegible]

[illegible] derived:public base[illegible]

[illegible]

[illegible] Occam’s Razor[illegible]

[illegible]

[illegible]

3. [illegible]

[illegible]“[illegible]”[illegible]“[illegible]”[illegible]“[illegible]”[illegible] oop[illegible]“[illegible]”[illegible] oop[illegible]

[illegible]

[illegible]

[illegible]

1.4.5 [illegible]

[illegible]

1. [illegible]

[illegible] “[illegible]” [illegible] “[illegible]” [illegible]

[illegible] “[illegible]” [illegible]

[illegible]

[illegible] main( ) [illegible] main( ) [illegible]

2. [illegible]

[illegible]

1) [illegible] “[illegible]” [illegible]

2) [illegible]

3) [illegible] //#[1] [illegible] // #[2] [illegible] Rich Text Format(RTF) [illegible]

4) [illegible]

5) [illegible]

[illegible]

1.4.6 [illegible]

[illegible]

[illegible] #include [illegible]

[illegible]

[illegible] main() [illegible] A、B、C [illegible] main() [illegible]

[illegible]

1.4.7 [illegible]

[illegible]

[illegible]

[illegible] “[illegible]” [illegible]

[illegible] “[illegible]” [illegible]

1.5 [illegible]

[illegible] 20 [illegible][1] [illegible]

[illegible]

---

[1] [illegible] Object Analysis and Design:Description of Methods, edited by Andrew T.F.Hutt of Object Management Group(OMG),john Wiley & Sons, 1994。

[illegible]

1.5.1 Booch

Booch[illegible] [1] [illegible] OOP[illegible]

1. [illegible]

[illegible]

2. [illegible]

[illegible] “ [illegible] ”[illegible]

3. [illegible] CRC [illegible]

[illegible] Class, Responsibility, Collaboration, CRC)[illegible]

4. [illegible]

[illegible]

5. [illegible]

[illegible] Booch[illegible] “ [illegible] ”[illegible] ——[illegible] /[illegible] CRC[illegible]

---

[1] [illegible] Object-Oriented Design with Applications by Grady Booch, Benjamin Cummings, 1991.[illegible] C++[illegible]

1.5.2 □□□□□□□□RDD□

□□□□[1]□□CRC□□□Booch□□□□□□□□——□□□□——□□□□□□□□□□RDD□□□□□□□□□——□□□□——□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□RDD□□□□□□

1) □□□□□□ □□□□□□□□□□□□□□□□

2) □□□□ □□□□□□□□□□□□□□□□□□□

3) □□□□□□□ □□□□□□□□□□□□□□□□□

4) □□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

1.5.3 □□□□□□□OMT□

□□□□□□[2]□OMT□□□□□□□□□□□□□□□Booch□□□□□□□□□□□□□□□□□□□□□□□□□□□□RDD□□□□□□□□□□□□□□□□□□□OMT□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

1) □□□□□“□□”□□□□□□ □□□□□□□□□Booch□□□RDD□□□□□□□□□□□□□□□□□□□

2) □□□□□“□□”□□□□□□ □□□

3) □□□□□“□□”□□□□□□□ □□

1.6 □□OOP□□□□□□□□

□□□□□□□□OOP□□□□□□□□□□□□“□□□□□□□□□□□□□□□□□□□□□□□OOP□”□□“□□□□”□□C□□C++□□□

1.6.1 □□□□OOP

□□OOP□C++□□□□□□□□□□□□□

1. □□

□□□□□□□□□□□□□□□□□□□□C□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□6□9□□□□□□□□□□□□□□□□□□□□□□□□□

---

[1] □□Designing Object-Oriented Software by Rebecca Wirfs-Brock et al., Prentice Hall, 1990□

[2] □□Object-Oriented Modeling and Design by James Rumbaugh et al., Prentice Hall,1991□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ C++□□□□□□□□□□□□□□

□□□、□□□□□□□□□□□□□□□□□□□□

2. □□□□□

□□□ OOP□□□ C++□□□□□□

3. □□□□□□□

□□[1]□□□□□□

4. □□□□□□□□

□□□ C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ main()□□ OOP□ C++□□□□□□□□

5. □□□ C++□□□□□□□□□

□□□ C++□□ C□□□ C++□□ C++□ OOP□□□

1.6.2 □□□□

□□ C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ C□□□□□□□□□□□□□□□□□□□□□□□□□□ C++□□□□□ OOP□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ C++□□□□□□□□□□□□□□□□□

1. □□□□

□□□□□□□□ C++□□□□□□□□□□□□□□□□□□□□□□□□□□、□□

---

[1] □□ Camma et al., □□□□□

[illegible] C++[illegible]

2. [illegible]

[illegible]“ OOP[illegible]” [illegible]“ [illegible]”[illegible] OOP[illegible] C++[illegible] C++[illegible]

[illegible] C[illegible] C++[illegible] ± 10%[illegible] C++[illegible] C[illegible] C++ [illegible] C[illegible]

[illegible] C[illegible] C++[illegible] C[illegible] C++[illegible] C[illegible] C++[illegible]

3. [illegible]

[illegible] OOP[illegible] C++[illegible] OOP[illegible]

1.7 [illegible]

[illegible] C++[illegible] OOP[illegible] C++[illegible] OOP[illegible] OOP[illegible] OOP[illegible] C++[illegible]

OOP[illegible] C++[illegible] C++[illegible] C++[illegible] C++[illegible]

[illegible]

[illegible] 2[illegible] [illegible]

C++[illegible] C[illegible]

[illegible]

[illegible]

[illegible] LIB[illegible] LIB [illegible] UNIX[illegible] Windows[illegible] DDL[illegible] DDL[illegible]

[illegible] C++[illegible] C[illegible] C++[illegible]

2.1 [illegible]

[illegible]“[illegible]”[illegible]“[illegible]”[illegible]“[illegible]”[illegible]“[illegible]”[illegible]“[illegible]”[illegible]“[illegible]”[illegible]

[illegible] A[illegible] int A[illegible]

[illegible] extern[illegible] extern[illegible] extern[illegible]

 int f(float,char);
f
C++

```
/*: DECLARE.C -- Declaration/definition examples */
extern int i; /* Declaration without definition */
extern float f(float); /* Function declaration */

float b;  /* Declaration & definition */
float f(float a) {  /* Definition */
  return a + 1.0;
}

int i; /* Definition */
int h(int x) { /* Declaration & definition */
  return x + 1;
}

main() {
  b = 1.0;
  i = 2;
  f(b);
  h(i);
}
```

C
C++

“ grep” “ awk”

2.2 C

C
C
struct
struct

C struct struct

stash

```
/*: LIB.H -- Header file: example C library */
/* Array-like entity created at run-time */

typedef struct STASHtag {
  int size;  /* Size of each space */
  int quantity; /* Number of storage spaces */
  int next; /* Next empty space */
  /* Dynamically allocated array of bytes: */
  unsigned char* storage;
} Stash;

void initialize(Stash* S, int Size);
void cleanup(Stash* S);
int add(Stash* S, void* element);
void* fetch(Stash* S, int index);
int count(Stash* S);
void inflate(Stash* S, int increase);
```

□□□□□□□□□□□□□□□□□□□□struct□□□□□□□□□□□□□□□□□□□□□struct□□□□□C□□□□□□□□□□□□□□□□□□□□□□□□typedef□□□□□□□□□□struct□□□□□□□□□□□□□□□□□□struct□□□□□□□

stash A, B, C;

□□□□□□□□□□□□ C □□□□□□□□□□ C □□□“□”C □□

storage□□□□□ unsigned char*□□□ C □□ stash□□□□□□□□□□□□□□□□□ void*□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□OBJ□ LIB□ DDL□□□□□

```
/*: LIB.C -- Implementation of
    example C library */
/* Declare structure and functions: */
#include "..\1\lib.h"
/* Error testing macros: */
#include <assert.h>
/* Dynamic memory allocation functions: */
#include <stdlib.h>
#include <string.h> /* memcpy() */
#include <stdio.h>
void initialize(Stash* S, int Size) {
  S->size = Size;
  S->quantity = 0;
  S->storage = 0;
  S->next = 0;
```

□ □

```
}

void cleanup(Stash* S) {
  if(S->storage) {
       puts("freeing storage");
       free(S->storage);
  }
}

int add(Stash* S, void* element) {
  /* enough space left? */
  if(S->next >= S->quantity)
    inflate(S, 100);
  /* Copy element into storage,
  starting at next empty space: */
  memcpy(&(S->storage[S->next * S->size]),
        element, S->size);
  S->next++;
  return(S->next - 1); /* Index number */
}

void* fetch(Stash* S, int index) {
  if(index >= S->next || index < 0)
    return 0;  /* Not out of bounds? */
  /* Produce pointer to desired element: */
  return &(S->storage[index * S->size]);
}

int count(Stash* S) {
  /* Number of elements in stash */
  return S->next;
}

void inflate(Stash* S, int increase) {
  void* v =
    realloc(S->storage,
           (S->quantity + increase)
           * S->size);
  /* Was it successful? */
  assert(v);
  S->storage = v;
  S->quantity += increase;
}
```

□ □ □ □ □ #include □ #include □ □ □ □ □

initialize( )□□□ struct stash □□□□□□□□□□□□□□□□□□□□□□□□ storage□□□□□□□□ size □□□□□□□□□□□□□□□□□□□

add( )□□□ stash□□ inflate( )□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ void*□□ C □□□ memcpy( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□ memcpy( )□□□□□□□□□□□□□□□□□□□□

&(S->storage[S->next * S->size])

□□□□□□□□□□□ next□□□ &□

memcpy( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ next□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ fetch( )□□□□□□□□□□□

fetch( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ add( )□□□□□□□□

□□□□□□ C□□□ count( )□□□ struct stash□□□□□□ intStash□□□□□□ intStash.next □□□□□□□□□□□□□□□□□□□□□□□□ count(&intStash)□□□□□□□□□□□□□□□□□□□□□□□□□ stash□□□“□□”□□□□□□□□□□□□□□ struct□□□□ next□□□□□□□□□□□□□□ next□□

□□□□□□

□□□□□□□□□□□ stash□□□□□□□□□□□□□□□ storage□□□ 200□ airplane□□□□□□□ 20□□□□□□□□□□□□□□ C □□□□□□□□ malloc( )□ calloc( )□ realloc( )□ free( )□

inflate( )□□□□ realloc( )□ stash□□□□□□□□ realloc( )□□ initialize( )□□□□□□□□ realloc( )□□ assert( )□□□□□□□□□□□□□□□□□□□□□□□□□ malloc( )□ calloc( )□ realloc( )□□□□□□□

□□□ C □□□□□□□□□□□□□□□□□□□□□□□ free( )□□□ Microsoft Windows□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

[illegible]

assert( )[illegible] ASSERT.H[illegible] assert( )[illegible]“ [illegible]” [illegible]“ [illegible] C [illegible]” [illegible] 17[illegible] C++[illegible]

[illegible] free( )[illegible] free( )[illegible] malloc( )[illegible] calloc( )[illegible] realloc( )[illegible] cleanup( )[illegible]

[illegible] stash[illegible] int[illegible] 80[illegible]

```
/*: LIBTESTC.C -- Test demonstration library */
#include "..\1\lib.h"
#include <stdio.h>
#include <assert.h>
#define BUFSIZE 80

main() {
  Stash intStash, stringStash;
  int i;
  FILE* file;
  char buf[BUFSIZE];
  char* cp;
/* .... */
initialize(&intStash, sizeof(int));
for(i = 0; i < 100; i++)
  add(&intStash, &i);
/* Holds 80-character strings: */
initialize(&stringStash,
           sizeof(char) * BUFSIZE);
file = fopen("LIBTESTC.C", "r");
assert(file);
while(fgets(buf, BUFSIZE, file))
  add(&stringStash, buf);
fclose(file);

for(i = 0; i < count(&intStash); i++)
  printf("fetch(&intStash, %d) = %d\n", i,
         *(int*)fetch(&intStash, i));
  i = 0;
  while((cp = fetch(&stringStash, i++)) != 0)
    printf("fetch(&stringStash, %d) = %s",
           i - 1, cp);
  putchar('\n');
  cleanup(&intStash);
  cleanup(&stringStash);
}
```

□ main( )□ □□□□□□□□□□□□□□□□ stash□ □□□□□□□□□□□□□□□ □□ □□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□“ □□□□□□□□□□□ □□□□□” □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□

intStash□ □□□□□ stringStash□ □□□□□□□□□□□□□□□□□□□□□ LIBTEST.C □□□□□□□ stringStash □□□□□□□□□□□□□□□□ C □□□□□ □□□□□□□□□□□□ stash□ □□□□□□□□□ fopen( )□ □□□□□□ FILE struct□ □□□ □□ FILE struct□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□ fgets( )□□ fclose( )□ □□□□□□□□□□□□□ FILE struct□ □□□□□□□□□□□ □□□□□□□ struct□ □□□□□ C □□□□□□□□□□□□

□□□□□□ stash□ □□□□□□□□□□□ intStach□ □□□□□ for□ □□□ count( )□ □ □□□□□ stringStash□ □□□□□ while□ □□□□ fetch( )□ □□□□□□□□□ ,□□□□□ □□

□□□□□□□ C □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□ C □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □ C□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□ int□ □□□□□□□□□ int□ □□□□□□□□□ □□□□□□□□□□□□

□□□□□□ LIB.H □□□□□□□ stash □□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□ C □□□□□

□□□□□ C□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□ □□ □□□□□□□□□□□□□□□□□□□□□□□□□□ void foo(float)□ □□□□□□□ □□□□□□□□□□□□□□□□□□ int□ □□ float□ □□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□ .o □ .obj □□□□□□□□ □□ □□□□□□□□□□□ LIBTEST.C □□□□□□□□ initialize( ) □ fetch( )□ (□□□□ □□□□□□□□□□□□□)□ □□□□□□□□ LIB.C □□□□ ,□□□□ LIBTEST.C □□□ □□

[illegible]

[illegible] C [illegible] foo(int) [illegible] foo(float) [illegible] _foo [illegible] _foo [illegible] foo( ) [illegible] int [illegible] foo( ) [illegible] float [illegible] float [illegible]

2.3 [illegible]

[illegible] cpp [illegible]:

```
cpp libtest.c lib.c
```

[illegible]

[illegible] UNIX [illegible] make [illegible] make [illegible] make [illegible] [1] [illegible] make [illegible]

make [illegible] makefile [illegible] makefile [illegible] makefile [illegible]

[illegible]

[illegible] C [illegible] .h [illegible] .c [illegible] C++ [illegible] Unix [illegible] .H [illegible] .C [illegible] MS-DOS [illegible] DOS C++ [illegible] .hxx [illegible] .cxx [illegible] C [illegible] C++ [illegible] .cpp [illegible] .h [illegible]

2.4 [illegible]

[illegible] stash [illegible] C [illegible]

---

[1] [illegible] C++ Inside & Out, (Osborne/McGraw-Hill,1993)[illegible]

[illegible]

[illegible] C [illegible] C [illegible]

[illegible] initialize( )[illegible] cleanup( )[illegible] C [illegible] C [illegible]

[illegible] C [illegible] initialize( )[illegible] cleanup( )[illegible] stash_initialize( )[illegible] stash_cleanup( )[illegible] “ [illegible] ” [illegible] struct [illegible] struct[illegible]

[illegible] C++[illegible] struct [illegible] struct[illegible] struct [illegible]

2.5 [illegible]

C++[illegible] “ [illegible] ”[illegible] stash [illegible]

```
//: LIBCPP.H -- C library converted to C++

struct stash {
  int size;  // Size of each space
  int quantity; // Number of storage spaces
  int next; // Next empty space
   // Dynamically allocated array of bytes:
  unsigned char* storage;
  // Functions!
  void initialize(int Size);
  void cleanup();
  int add(void* element);
  void* fetch(int index);
  int count();
  void inflate(int increase);
};
```

[illegible] //[illegible] C [illegible] C [illegible] C++[illegible] //[illegible]

[illegible] typedef[illegible] C++[illegible] typedef[illegible] int[illegible] char[illegible] float[illegible] double[illegible] stash [illegible]

[illegible] struct[illegible] C [illegible] C++[illegible]

[illegible] C [illegible]

```
stash A, B, C;
```

[illegible] add( )[illegible]

[illegible] C [illegible] stash_initialize( )[illegible] stash_cleanup( )[illegible] “ [illegible] ”[illegible] struct [illegible] stash [illegible] initialize( ) [illegible] initialize( ) [illegible] ——[illegible] initialize( )[illegible] struct stash [illegible] struct[illegible] C++[illegible] ::[illegible] struct [illegible] initialize( )[illegible] stash[illegible] stash::initialize(int Size, int Quantity)[illegible] stash [illegible] C++[illegible]

```
//: LIBCPP.CPP -- C library converted to C++
// Declare structure and functions:
#include "..\1\libcpp.h"
#include <assert.h> // Error testing macros
#include <stdlib.h> // Dynamic memory
#include <string.h> // memcpy()
#include <stdio.h>

void stash::initialize(int Size) {
  size = Size;
  quantity = 0;
  storage = 0;
  next = 0;
}

void stash::cleanup() {
  if(storage) {
    puts("freeing storage");
    free(storage);
  }
}

int stash::add(void* element) {
  if(next >= quantity) // Enough space left?
    inflate(100);
  // Copy element into storage,
  // starting at next empty space:
  memcpy(&(storage[next * size]),
         element, size);
  next++;
  return(next - 1); // Index number
}

void* stash::fetch(int index) {
  if(index >= next || index < 0)
    return 0;  // Not out of bounds?
  // Produce pointer to desired element:
  return &(storage[index * size]);
}

int stash::count() {
  return next; // Number of elements in stash
}

void stash::inflate(int increase) {
  void* v =
    realloc(storage, (quantity+increase)*size);
  assert(v);  // Was it successful?
  storage = (unsigned char*)v;
  quantity += increase;
}
```

[illegible] C++ [illegible] C++ [illegible]

[illegible]

[illegible] C [illegible] struct [illegible] s->size = size [illegible] size = size [illegible] s-> [illegible] C++ [illegible] “ [illegible] ” [illegible] this->size = Size;

this

S->storage = v;

calloc( )realloc( )

unsigned char*

//: LIBTEST.CPP -- Test of C++ library
#include "..\1\libcpp.h"
#include <stdio.h>
#include <assert.h>
#define BUFSIZE 80

main() {
  stash intStash, stringStash;
  int i;
  FILE* file;
  char buf[BUFSIZE];
```
```

```
  char* cp;
  // ....
  intStash.initialize(sizeof(int));
  for(i = 0; i < 100; i++)
    intStash.add(&i);
  // Holds 80-character strings:
  stringStash.initialize(sizeof(char)*BUFSIZE);
  file = fopen("LIBTEST.CPP", "r");
  assert(file);
  while(fgets(buf, BUFSIZE, file))
    stringStash.add(buf);
  fclose(file);

  for(i = 0; i < intStash.count(); i++)
    printf("intStash.fetch(%d) = %d\n", i,
           *(int*)intStash.fetch(i));

  i = 0;
  while(
    (cp = (char*)stringStash.fetch(i++))!=0)
      printf("stringStash.fetch(%d) = %s",
             i - 1, cp);
  putchar('\n');
  intStash.cleanup();
  stringStash.cleanup();
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“ .”□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□this □□□ C++□□□□ intStash.initialize(sizeof(int), 100) □□ stash_initialize(&intStash, sizeof(int), 100)□□□□□□□□□□□□□□□□□□□□□□□□□□□□ C++□□□ cfront□□□ AT&T□□□□□□□□ C□□□□□□□ C □□□□□□□□□□□□□ cfront □□ C++□□□□□□ C □□□□□□□□□□□□□ C++ □□□□□□

□□□□□□□□□□□□□□□□□□□□

while(cp = (char*)stringStash.fetch(i++))

□□□□□ C++ □□□□□□□□□□□□□□□

2.6 □□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ C++ □□□□□□□□□□□□□□□□□□□□□□□□□□□ C □□□

[illegible] C[illegible] struct [illegible]

“[illegible]”[illegible]“[illegible]”[illegible] OOP[illegible] OOP[illegible] Simula-67[illegible] 1967[illegible] Scandinavia[illegible] Simula[illegible] Simula [illegible]“[illegible]”[illegible] class [illegible] C++ [illegible]“[illegible]”[illegible]

[illegible] object-based[illegible] C++ [illegible]

2.7 [illegible]

[illegible][1][illegible] float[illegible] float [illegible] int [illegible]

stash [illegible] add( )[illegible] fetch( )[illegible] inflate( )[illegible] stash S[illegible] stash [illegible] float f [illegible] float [illegible] stash [illegible]——[illegible] stash[illegible] abstract dada type[illegible] C++[illegible] stash[illegible] stash [illegible]

[illegible] object.member_function(arglist)[illegible]“[illegible]”[illegible]“[illegible]”[illegible] stash S[illegible] S.add(&i)“[illegible] S”[illegible]“[illegible] add( )”[illegible]“[illegible]”[illegible] C++ [illegible]

2.8 [illegible]

[illegible] C [illegible] C [illegible]“[illegible]”[illegible]“[illegible] C[illegible] struct[illegible]”[illegible] C struct [illegible] C++ [illegible] C [illegible] C++ [illegible] C[illegible]

---

[1] [illegible]

[illegible] struct [illegible] 14[illegible] 16[illegible] “[illegible]” [illegible]

[illegible] sizeof[illegible] struct [illegible]

```
//: SIZEOF.CPP -- Sizes of structs
#include <stdio.h>
#include "..\1\lib.h"
#include "..\1\libcpp.h"
struct A {
  int I[100];
};

struct B {
  void f();
};

void B::f() {}

main() {
  printf("sizeof struct A = %d bytes\n",
         sizeof(A));
  printf("sizeof struct B = %d bytes\n",
         sizeof(B));
  printf("sizeof Stash in C = %d bytes\n",
         sizeof(Stash));
  printf("sizeof stash in C++ = %d bytes\n",
         sizeof(stash));
}
```

[illegible] 200[illegible] int [illegible] struct B [illegible] struct[illegible] C [illegible] C++ [illegible] struct[illegible] printf ()[illegible]

[illegible] sizeof [illegible] C++ [illegible] C [illegible] C++ [illegible]

2.9 [illegible]

[illegible] C [illegible] C[illegible] C++ [illegible] CPP [illegible]

[illegible]

[illegible]“[illegible]”[illegible] CPP [illegible] “[illegible]” [illegible]

[illegible]

[illegible]

[illegible] “[illegible]” [illegible]

[illegible] “[illegible]” [illegible]

[illegible] C [illegible] C++ [illegible] C++ [illegible]

[illegible] C++ [illegible]

[illegible] FOO.H [illegible] “[illegible]” [illegible] FOO.H [illegible]

```
#ifndef FOO_H_
#define FOO_H_
// Rest of header here ...
#endif // FOO_H_
```

[illegible] C [illegible]

[illegible]

[illegible] C++ [illegible]

[illegible] main( ) [illegible]

2.10 [illegible]

[illegible]

[illegible]

```
//: NESTED.H -- Nested struct in linked list
#ifndef NESTED_H_
#define NESTED_H_

struct stack {
  struct link {
    void* data;
    link* next;
    void initialize(void* Data, link* Next);
  } * head;
  void initialize();
  void push(void* Data);
  void* peek();
  void* pop();
  void cleanup();
};
#endif // NESTED_H_
```

[illegible] struct [illegible] link[illegible] link [illegible] link [illegible] next [illegible]

[illegible] head [illegible] struct link [illegible] link* head[illegible] C [illegible]

[illegible] initialize( ) [illegible] stack [illegible] initialize( ) [illegible] cleanup( ) [illegible] push( ) [illegible] pop( ) ,[illegible] data[illegible] data [illegible] peek( )[illegible] data [illegible]

cleanup [illegible] data [illegible]

[illegible]

```
//: NESTED.CPP -- Linked list with nesting
#include <stdlib.h>
#include <assert.h>
#include "nested.h"

void stack::link::initialize(
    void* Data, link* Next) {
  data = Data;
  next = Next;
}

void stack::initialize() { head = 0; }
void stack::push(void* Data) {
  link* newlink = (link*)malloc(sizeof(link));
  assert(newlink);
  newlink->initialize(Data, head);
  head = newlink;
}

void* stack::peek() { return head->data; }

void* stack::pop() {
  if(head == 0) return 0;
  void* result = head->data;
  link* oldHead = head;
  head = head->next;
  free(oldHead);
  return result;
}

void stack::cleanup() {
  link* cursor = head;
  while(head) {
    cursor = cursor->next;
    free(head->data); // Assumes a malloc!
    free(head);
   head = cursor;
  }
}
```

□□ struct □□□□ stack::link::initialize( ) □□

stack::initialize( ) □□□ head □□□□□□□□□□□□□□□□□□

stack::push( ) □□□□□□□□□□□□□□□□ stack □□□□□□□□□□□□□□□□□□□□□□□□□□□ malloc( ) □ link □□□□ link□ □□□□□□□□□ initialize( ) □□□□□ link □□□□□□□□□□□ next □□□□□□ head□ □ head □□□ link □□□□□□□□ link □□□□□□□□□

stack::pop( )□ □□□□□□□□ data □□□□□□□ head □□□□□□□□□□□□□ stack::cleanup( )□ □ cursor □□□□□□□□ free( )□ □□□ link□ data□ link □□□

□□□□□□□□□□□□

```
//: NESTEST.CPP -- Test of nested linked list
#include "..\1\nested.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
```

```
main(int argc, char** argv) {
  stack textlines;
  FILE* file;
  char* s;
  #define BUFSIZE 100
  char buf[BUFSIZE];
  assert(argc == 2); // File name is argument
  textlines.initialize();
  file = fopen(argv[1], "r");
  assert(file);
  // Read file and store lines in the stack:
  while(fgets(buf, BUFSIZE, file)) {
    char* string =(char*)malloc(strlen(buf)+1);
    assert(string);
    strcpy(string, buf);
    textlines.push(string);
  }
  // Pop the lines from the stack and print them:
  while((s = (char*)textlines.pop()) != 0) {
    printf("%s", s); free(s); }
  textlines.cleanup();
}
```

[illegible]

[illegible]

[illegible]“[illegible]”[illegible] A [illegible] A [illegible]

```
//: SCOPERES.CPP -- Global scope resolution
int A;
void f() {}

struct S {
  int A;
  void f();
};

void S::f() {
  ::f();  // Would be recursive otherwise!
  ::A++;  // Select the global A
  A--;    // The A at struct scope
}

main() {}
```

□□□ S::f( )□□□□□□□□□□□□□□□□□ f( )□ A□□□□□□

2.11 □□

□□□□□□□□□□□□□□□C++□□□

□□C++□□□□□□□□□□□□□□□□□□□□□□ struct □□□

2.12 □□

1) □□□□ struct □□

2) □□□□□□□□□□□□□□□□□□□□□□□□□□□□□

3) □□

4) □□

5) C++□□□□□□□□ struct □□□□ typedef □□□□□□□□□□□□□□□□□□□□□□□□□□□□□

China-pub.com [illegible]

[illegible] 3[illegible] [illegible]

[illegible] C[illegible] C++[illegible]

[illegible] struct[illegible] struct[illegible] struct[illegible] typedef[illegible]

[illegible] ——[illegible] C++[illegible] struct[illegible]

3.1 [illegible]

[illegible]

[illegible] C[illegible] struct[illegible] struct[illegible] initialize()[illegible] cleanup()[illegible] struct[illegible] C[illegible]

[illegible]

[illegible] struct[illegible] stack[illegible] malloc() [illegible]

3.2 C++[illegible]

C++[illegible] struct[illegible] public[illegible] private[illegible] protected[illegible] struct[illegible]

public[illegible] public[illegible] struct[illegible] struct[illegible]

```
//: PUBLIC.CPP -- Public is just like C struct

struct A {
  int i;
  char j;
  float f;
  void foo();
};

void A::foo() {}

struct B {
public:
  int i;
  char j;
  float f;
  void foo();
};

void B::foo() {}
```

```
main() {}
```

private□□private□□ struct B□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
//: PRIVATE.CPP -- Setting the boundary

struct B {
private:
  char j;
  float f;
public:
  int i;
  void foo();
};

void B::foo() {
  i = 0;
  j = '0';
  f = 0.0;
};

main() {
  B b;
  b.i = 1;    // OK, public
//!  b.j = '1';  // Illegal, private
//!  b.f = 1.0;  // Illegal, private
}
```

[illegible] foo()[illegible] B[illegible] main()[illegible] struct [illegible] struct[illegible]

[illegible]

[illegible] (protected)

[illegible] protected[illegible] protected[illegible] private[illegible] protected[illegible] private[illegible] 13[illegible]

3.3 [illegible]

[illegible] struct[illegible] struct[illegible] “[illegible]”[illegible]

[illegible] “[illegible]”[illegible] “[illegible]”[illegible]

[illegible] struct[illegible] struct[illegible]

```
//: FRIEND.CPP -- Friend allows special access

struct X; // Declaration (incomplete type spec)

struct Y {
  void f(X*);
};

struct X { // Definition
private:
  int i;
public:
  void initialize();
  friend void g(X*, int); // Global friend
  friend void Y::f(X*);  // Struct member friend
  friend struct Z; // Entire struct is a friend
  friend void h();
};

void X::initialize() { i = 0; }

void g(X* x, int i) { x->i = i; }

void Y::f(X* x) { x->i = 47; }
```

```
struct Z {
private:
  int j;
public:
  void initialize();
  void g(X* x);
};

void Z::initialize() { j = 99; }

void Z::g(X* x) { x->i += j; }

void h() {
  X x;
  x.i = 100; // Direct data manipulation
}

main() {
  X x;
  Z z;
  z.g(&x);
}
```

struct Y□□□□□□□ f( )□□□□□ X□□□□□□□□□□□□□□□□ C++□□□□□□□□□□□□□□□□□□□□□□ struct Y□□□□□□□ Y :: f(X*)□□□□ struct X□□□□□□□□□ □□ Y :: f(X*)□□□□□ struct X□□□□□□□

□□□□□□□□□□ Y :: f(X*)□□□□□ X□□ X□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ Y :: g(X)□□□□

□□□□ X□□□□□□□□□□□□□□ Y :: f(X*)□□□□□□□□□□□□□□□□□□ struct X□□□□□□□□□□□□□□□□□□□□□□ X□ struct,□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□ struct X□□ Y :: f(X*)□□□□□□□□□□□□□□□□□□□□□□□□□□□ Y□□□

□□□□□□□□□□□□□□□□□□□□□□□□ g( )□□□□□□□□ g( )□□□□□□□□□□□□□□□□□□□□□ friend□□□□□□□□□□□□□□□ struct□□□□□□□□□□□ struct□□□□ :friend struct Z□□□□□□□□□□□□□□□□ struct□□□□□□□□

3.3.1 □□□□

□□□□□ struct□□ struct□□□□□□□□□□□□□□□□□□□

struct□□□□□□friend□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
//: NESTFRND.CPP -- Nested friends
#include <stdio.h>
#include <string.h> // memset()
#define SZ 20

struct holder {
private:
  int a[SZ];
public:
  void initialize();
  struct pointer {
  private:
    holder* h;
    int* p;
  public:
    void initialize(holder* H);
    // Move around in the array:
    void next();
    void previous();
    void top();
    void end();
    // Access values:
    int read();
    void set(int i);
  };
  friend holder::pointer;
};

void holder::initialize() {
 memset(a, 0, SZ * sizeof(int));
}

void holder::pointer::initialize(holder* H) {
  h = H;
  p = h->a;
}

void holder::pointer::next() {
  if(p < &(h->a[SZ - 1])) p++;
}

void holder::pointer::previous() {
  if(p > &(h->a[0])) p--;
}
```

```
void holder::pointer::top() {
  p = &(h->a[0]);
}

void holder::pointer::end() {
  p = &(h->a[SZ - 1]);
}

int holder::pointer::read() {
  return *p;
}

void holder::pointer::set(int i) {
  *p = i;
}

main() {
  holder h;
  holder::pointer hp, hp2;
  int i;

  h.initialize();
  hp.initialize(&h);
  hp2.initialize(&h);
  for(i = 0; i < SZ; i++) {
    hp.set(i);
    hp.next();
  }
  hp.top();
  hp2.end();
  for(i = 0; i < SZ; i++) {
    printf("hp = %d, hp2 = %d\n",
           hp.read(), hp2.read());
    hp.next();
    hp2.previous();
  }
}
```

struct holder□□□□□□□□□□□ pointer,□□□□□□ pointer□□□□□□□□□□ pointer□ holder□□□□□□□□□□□□□□ struct□□□□□□□□□ pointer□□□□□□□□□□□□□□□□□□□□ holder□□□□□□□□□

friend holder : :pointer ;

□□□□□ struct□□□□□□□□□□□□□□□□□□□□ pointer□□□□□

□□ pointer□□ holder□□□□□□□□□□□□ main( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ pointer□ C□□□□□□□□□□□□□□□□□□□□□□□□□□ holder□□□□

3.3.2 [illegible]

[illegible] friend[illegible]

C++[illegible] friend[illegible] C++[illegible]

3.4 [illegible]

[illegible] 2[illegible] C[illegible] struct[illegible] C++[illegible] struct[illegible] C++[illegible] C struct [illegible] C[illegible] struct[illegible] C++[illegible]

[illegible] C++[illegible] “[illegible]”[illegible] C[illegible] “[illegible]” [illegible]

[illegible] C++[illegible]

[illegible] struct[illegible] struct[illegible] C[illegible] C++[illegible]

[illegible] struct[illegible]

3.5 [illegible]

[illegible] struct[illegible]

[illegible] private[illegible]

[illegible] C[illegible] struct[illegible]

[illegible] Simula-67[illegible] class[illegible] Stroustrup[illegible] C++[illegible]

□□□□□□ C□□□□□□□ struct□□□□□□□□□□□□□□□□

□□ class□ C++□□□□□□□□□□□□□□□□□□□□□ struct□□□□□□□□□ □□□□ class□□□□□□□□□□□□ struct□□□□□□□ public□□□□□□□□□□□ □□□□□□□□□

```
//: CLASS.CPP -- Similarity of struct and class

struct A {
private:
  int i, j, k;
public:
  int f();
  void g();
};

int A::f() { return i + j + k; }

void A::g() { i = j = k = 0; }

// Identical results are produced with:

class B {
  int i, j, k;
public:
  int f();
  void g();
};

int B::f() { return i + j + k; }

void B::g() { i = j = k = 0; }

main() {}
```

□ C++□□ class□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□ class□□□□□□□□□□□□□□□□□□□□□□ Stroustrup□□□□ struct□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□ struct□□□□□□□□□□□□□□□□ public□□□□□ “ □ □□□□ ” □□□□□

```
class X {
public:
  void interface_function();
private:
  void private_function();
  int internal_representation;
};
```

China-pub.com [illegible]

[illegible]

[illegible]

```
class X {
  void private_function();
  int internal_representation;
public:
  void interface_function();
};
```

[illegible]

```
class Y {
public:
  void f();
private:
  int mX;  // "self-mangled" name
};
```

[illegible] mX[illegible] Y[illegible] x[illegible] m[illegible]

3.5.1 [illegible] stash

[illegible] 2[illegible]

```
//: STASH.H -- Converted to use access control
#ifndef STASH_H_
#define STASH_H_
class stash {
  int size;  // Size of each space
  int quantity; // Number of storage spaces
  int next; // Next empty space
  // Dynamically allocated array of bytes:
  unsigned char* storage;
  void inflate(int increase);
public:
  void initialize(int Size);
  void cleanup();
  int add(void* element);
  void* fetch(int index);
  int count();
};
#endif // STASH_H_
```

inflate()□ □□□□□□□□□□□□ add()□ □□□□□□□□□□□□□□□□□□□ □□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

3.5.2 □□□□□□□□ stack

□□□□□□□□□□□ stack□ □□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□ stack□ □□□□□

```
//: STACK.H -- Nested structs via linked list
#ifndef STACK_H_
#define STACK_H_

class stack {
  struct link {
    void* data;
    link* next;
    void initialize(void* Data, link* Next);
  } * head;
public:
  void initialize();
  void push(void* Data);
  void* peek();
  void* pop();
  void cleanup();
};
#endif // STACK_H_
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□ □□□ □□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

3.6 □□□□ handle classes□

C++□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□ □□□□□□□□□□□□□□□□□□□□□□□□□ C++□ □□□□□□□□□□□□ □□□ □□□ □□□□□□□□□□□

3.6.1 □□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

[illegible] ([illegible])[illegible] "[illegible]" [illegible]

3.6.2 [illegible]

[illegible]

[illegible] handle classes[illegible] "Cheshire Cat"[1][illegible] "smile"[illegible]

[illegible]

```
//: HANDLE.H -- Handle classes
#ifndef HANDLE_H_
#define HANDLE_H_

class handle {
  struct cheshire; // Class declaration only
  cheshire* smile;
public:
  void initialize();
  void cleanup();
  int read();
  void change(int);
};
#endif // HANDLE_H_
```

[illegible]

```
struct cheshire;
```

[illegible] cheshire [illegible]

---

[1] [illegible] John Carolan[illegible] Lewis Carroll[illegible] C++[illegible]

```
//: HANDLE.CPP -- Handle implementation
#include "..\2\handle.h"
#include <stdlib.h>
#include <assert.h>

// Define handle's implementation:
struct handle::cheshire {
  int i;
};

void handle::initialize() {
  smile = (cheshire*)malloc(sizeof(cheshire));
  assert(smile);
  smile->i = 0;
}

void handle::cleanup() {
  free(smile);
}

int handle::read() {
  return smile->i;
}

void handle::change(int x) {
  smile->i = x;
}
```

cheshire □

```
struct handle::cheshire {
```

□ handle::initialize()□ □ □ cheshire struct□ □ □ □ □ □ [1]□ □ handle::cleanup()□ HANDLE.CPP□ cheshire□ □ □ □ □ □ □ □ □ □ □ □ □ □ HANDLE.CPP□ □ □ □ □ □ □ □ □ □ □ □

□ □ □ handle□ □

```
//: USEHANDL.CPP -- Use the handle class
#include "..\2\handle.h"

main() {
  handle u;
  u.initialize();
  u.read();
```

---

[1] □ □ 12□ new□ □ □ □ □ □ □ □

```
    u.change(1);
    u.cleanup();
  }
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

3.7 □□

□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□ □□□ □□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□ □□□□□□□□□□□□□□□□□□□□□□□□□

3.8 □□

1) □□□□□□□□ public□ private □ protected□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□

2) □□□□□□□□□□ friend□□□□□□□□□□□□

3) □□ HANDLE.CPP□□ cheshire□□□□□□□□□□□□□□□□□□□ USEHANDL.CPP□

[illegible]4[illegible] [illegible]

[illegible]2[illegible] C[illegible] struct[illegible]

[illegible] 3 [illegible]

[illegible] C[illegible] C++[illegible]

[illegible] C++[illegible] C[illegible] C++[illegible]

[illegible] C[illegible] struct [illegible] struct[illegible] C[illegible] struct[illegible]

[illegible] C++[illegible] C++[illegible]

4.1 [illegible]

[illegible] stash[illegible] stack[illegible] initialize()[illegible] C++[illegible]

[illegible] Stroustrup[illegible]

[illegible]

```
class X {
  int i;
public:
  X();  // constructor
};
```

[illegible]

```
void f() {
  X a;
  // ...
}
```

[illegible] a[illegible] a[illegible] a[illegible] X::X()[illegible]

[illegible] tree[illegible]

```
tree t(12); // 12[illegible]
```

[illegible] tree(int)[illegible]

[illegible] initialize()[illegible] C++[illegible]

[illegible] void[illegible]

4.2 [illegible]

[illegible] C[illegible] ?[illegible] “[illegible]” [illegible] ?[illegible] “[illegible]”[illegible] C++[illegible]

[illegible] ~[illegible]

```
class Y {
public:
  ~Y();
};
```

□□ goto□□□□□□□□□□□□□ C □□□□□□□ goto□ C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ goto□□□□□□□□ C□□□□□ setjmp() □ longjmp()□□□

□□□□□□□□□□□□□□□□□□□□

```
//: CONSTR1.CPP -- Constructors & destructors
#include <stdio.h>

class tree {
  int height;
public:
  tree(int initialHeight);  // Constructor
  ~tree();  // Destructor
  void grow(int years);
  void printsize();
};

tree::tree(int initialHeight) {
  height = initialHeight;
}

tree::~tree() {
  puts("inside tree destructor");
  printsize();
}

void tree::grow(int years) {
  height += years;
}

void tree::printsize() {
  printf("tree height is %d\n", height);
}

main() {
  puts("before opening brace");
  {
    tree t(12);
    puts("after tree creation");
    t.printsize();
    t.grow(4);
    puts("before closing brace");
  }
  puts("after closing brace");
}
```

[illegible]

```
before opening brace
after tree creation
tree height is 12
before closing brace
inside tree destructor
tree height is 16
after closing brace
```

[illegible]

4.3 [illegible]

[illegible] C[illegible] Pascal[illegible]“[illegible]”[illegible]

[illegible] C++[illegible] C[illegible] C++[illegible] C[illegible] C[illegible]

[illegible] C++[illegible] C++[illegible]

```
//: DEFINIT.CPP -- Defining variables anywhere
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>

class G {
  int i;
public:
  G(int I);
};

G::G(int I) { i = I; }
```

```
main() {
  #define SZ 100
  char buf[SZ];
  printf("initialization value? ");
  int retval = (int)gets(buf);
  assert(retval);
  int x = atoi(buf);
  int y = x + 3;
  G g(y);
}
```

[illegible] buf[illegible] x[illegible] y[illegible] g[illegible] C[illegible]

4.3.1 for[illegible]

[illegible] C++[illegible] for[illegible] for[illegible]

```
for(int j = 0; j < 100; j++) {
    printf("j = %d\n", j);
}
for(int i = 0; i < 100; i++)
    printf("i = %d\n", i);
```

[illegible] C++[illegible]

[illegible] i[illegible] j[illegible] for[illegible] C[illegible] for[illegible] i[illegible] j[illegible] i_loop_counter[illegible]

[illegible] i[illegible]

```
for(int i = 0; i < 100; i++)
    printf("i = %d\n", i);
// ....
for(int i = 0; i < 100; i++){
    printf("i = %d\n", i);
}
```

[illegible] C++[illegible] for[illegible]

[illegible]

[illegible]

4.3.2 [illegible]

[illegible] C[illegible] switch[illegible] goto[illegible]

[illegible]

```
//: NOJUMP.CPP -- Can't jump past constructors

class X {
public:
  X() {}
};

void f(int i) {
  if(i < 10) {
   //! goto jump1; // Error: goto bypasses init
  }
  X x1;  // Constructor called here
 jump1:
  switch(i) {
    case 1 :
      X x2;  // Constructor called here
      break;
  //! case 2 : // Error: case bypasses init
      X x3;  // Constructor called here
      break;
  }
}

main() {}
```

[illegible] goto[illegible] switch[illegible]

[illegible] 12[illegible]

4.4 [illegible] stash

[illegible] initialize()[illegible] cleanup()[illegible] stash[illegible]

```
//: STASH3.H -- With constructors & destructors
#ifndef STASH3_H_
#define STASH3_H_

class stash {
  int size;  // Size of each space
  int quantity; // Number of storage spaces
  int next; // Next empty space
  // Dynamically allocated array of bytes:
  unsigned char* storage;
  void inflate(int increase);
public:
  stash(int Size);
  ~stash();
  int add(void* element);
  void* fetch(int index);
  int count();
};
#endif // STASH3_H_
```

[illegible] initialize()[illegible] cleanup()[illegible]

```
//: STASH3.CPP -- Constructors & destructors
#include "..\3\stash3.h"
#include <assert.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

stash::stash(int Size) {
  size = Size;
  quantity = 0;
  storage = 0;
  next = 0;
}

stash::~stash() {
  if(storage) {
    puts("freeing storage");
    free(storage);
  }
}

int stash::add(void* element) {
  if(next >= quantity) // Enough space left?
    inflate(100);
  // Copy element into storage,
  // starting at next empty space:
  memcpy(&(storage[next * size]),
         element, size);
  next++;
  return(next - 1); // Index number
}

void* stash::fetch(int index) {
  if(index >= next || index < 0)
    return 0;  // Not out of bounds?
  // Produce pointer to desired element:
  return &(storage[index * size]);
}

int stash::count() {
  return next; // Number of elements in stash
}

void stash::inflate(int increase) {
  void* v =
    realloc(storage, (quantity+increase)*size);
  assert(v);  // Was it successful?
  storage = (unsigned char*)v;
  quantity += increase;
}
```

□□□□□□□□□□□□□stash□□

```
//: STSHTST3.CPP -- Constructors & destructors
#include "..\3\stash3.h"
#include <stdio.h>
#include <assert.h>
#define BUFSIZE 80

main() {
  stash intStash(sizeof(int));
  for(int j = 0; j < 100; j++)
    intStash.add(&j);

  FILE* file = fopen("STASHTST.CPP", "r");
  assert(file);
// Holds 80-character strings:
stash stringStash(sizeof(char) * BUFSIZE);
char buf[BUFSIZE];
while(fgets(buf, BUFSIZE, file))
  stringStash.add(buf);
fclose(file);
  for(int k = 0; k < intStash.count(); k++)
    printf("intStash.fetch(%d) = %d\n", k,
            *(int*)intStash.fetch(k));

  for(int i = 0; i < stringStash.count(); i++)
    printf("stringStash.fetch(%d) = %s",
            i, (char*)stringStash.fetch(i++));
  putchar('\n');
}
```

□ □ □ cleanup()□ □ □ □ □ □ □ □ □ intStash□ stringStash□ □

4.5 □ □ □ □ □ □ □ □ □ □ □ □ stack

□ stack□ □□ □ □ □ □ □ □ □ □ □ □ □

```
//: STACK3.H -- With constructors/destructors
#ifndef STACK3_H_
#define STACK3_H_

class stack {
  struct link {
    void* data;
    link* next;
    void initialize(void* Data, link* Next);
  } * head;
public:
  stack();
  ~stack();
  void push(void* Data);
  void* peek();
  void* pop();
};
#endif // STACK3_H_
```

□ □ □ □ □ stack□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ link□ □

```
//: STACK3.CPP -- Constructors/destructors
#include <stdlib.h>
#include <assert.h>
```

```
#include "..\3\stack3.h"

void stack::link::initialize(
  void* Data, link* Next) {
  data = Data;
  next = Next;
}

stack::stack() { head = 0; }

void stack::push(void* Data) {
  // Can't use a constructor with malloc!
  link* newlink = (link*)malloc(sizeof(link));
  assert(newlink);
  newlink->initialize(Data, head);
  head = newlink;
}

void* stack::peek() { return head->data; }

void* stack::pop() {
  if(head == 0) return 0;
  void* result = head->data;
  link* oldHead = head;
  head = head->next;
  free(oldHead);
  return result;
}

stack::~stack() {
  link* cursor = head;
  while(head) {
    cursor = cursor->next;
    free(head->data); // Assumes malloc!
    free(head);
    head = cursor;
  }
}
```

link□ □ stack::push□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“ □□□ □□□□□□□□□□□□□□□□□□□□□□□□□□” □□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□ [1]□□□□□□□□□□ C++□ □□□□□□□□□□□

---

[1] □□

□□□□□□□□□□□□□□□□□□□□□□new□□□□□□□□□12□□□□□□□□□□□□□□C□□□□□□□□□□□□□□□□□□□□□□□□□stack□□□□□□□□□□□□□□□□□□□□□□□□□

```
//: STKTST3.CPP -- Constructors/destructors
#include "..\3\stack3.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

main(int argc, char** argv) {
  assert(argc == 2); // File name is argument
  FILE* file = fopen(argv[1], "r");
  assert(file);
  #define BUFSIZE 100
  char buf[BUFSIZE];
  stack textlines; // Constructor called here
  // Read file and store lines in the stack:
  while(fgets(buf, BUFSIZE, file)) {
    char* string =
      (char*)malloc(strlen(buf) + 1);
    assert(string);
    strcpy(string, buf);
    textlines.push(string);
  }
  // Pop lines from the stack and print them:
  char* s;
  while((s = (char*)textlines.pop()) != 0) {
    printf("%s", s); free(s); }
} // Destructor called here
```

textlines□□

4.6 □□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□struct□class□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□C++□□

int a[5]={1,2,3,4,5};

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

[illegible]

```
int b[6]={0};
```

[illegible] 0[illegible] for[illegible]“[illegible] 1[illegible]”[illegible] for[illegible]

[illegible]

```
int c[] = {1,2,3,4};
```

[illegible] sizeof c/sizeof *c([illegible])[illegible]

```
for(int i = 0; i< sizeof c / sizeof *c; i++)
  c[i]++;
```

struct[illegible] C[illegible] struct[illegible]

```
struct X {
  int i;
  float f;
  char c;
};
X x1 = {1,2.2,'c' };
```

[illegible] struct[illegible]

```
X x2[3] = {{1,1.1, 'a'},{2,2.2, 'b'}};
```

[illegible]

[illegible] struct[illegible] struct [illegible]

```
struct Y {
  float f;
  int i;
  Y(int A); // presumably assigned to i
};
```

[illegible]

```
Y y2[] = {Y(1),Y(2),Y(3)};
```

[illegible] struct[illegible] class,[illegible]

[illegible]

```
//: MULTIARG.CPP -- Multiple constructor arguments
// with aggregate initialization

class X {
  int i, j;
public:
  X(int I, int J) {
    i = I;
    j = J;
  }
};

main() {
  X xx[] = { X(1,2), X(3,4), X(5,6), X(7,8) };
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

4.7 □□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“ □□□ vanilla□ □□” ,□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□ Y□ □□□□□□□□□

Y y4[2] = {Y(1)}

□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□ Y□ □□□□□

Y y5[7]□

□□□□□□□□

Y y;

□□□

□□□□□□□□□□□□□□□□□□□□□□□ struct □ class□ □□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□

```
class Z {
  int i; // private
}; // no constructor

Z z,z2[10];
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□ C□ □□□□□□□□ C++□ □□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□

□□□ C++□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ C□ □□□□□□□□□□ C++□ □□□□□□□□□ C□ □□□□□ struct□ □□□□

[illegible]C++[illegible]

[illegible] C[illegible] C++[illegible] C[illegible] C++[illegible] C++[illegible] C[illegible] C++[illegible] C++[illegible] C[illegible]

4.8 [illegible]

[illegible]C++[illegible] Stroustrup[illegible] C++[illegible] C[illegible]

[illegible]——[illegible]

[illegible] C++[illegible]

4.9 [illegible]

1) [illegible]3[illegible] HANDLE.H,HANDLE.CPP[illegible] USEHANDL.CPP [illegible]

2) [illegible]

3) [illegible]

4) [illegible]

[illegible]

[illegible] 5 [illegible]

[illegible]

[illegible] —— [illegible] —— [illegible] “ [illegible] ” [illegible] “ [illegible] ” [illegible] —— [illegible]

[illegible] print_int() [illegible] print_char() [illegible] print_float(), [illegible]

[illegible] C++ [illegible]

[illegible]

[illegible] C++ [illegible] f(“ hello”),f(“ hi” ,1) [illegible] f(“ howdy” ,2,‘ c’) [illegible]

[illegible]

5.1 [illegible]

[illegible] 2 [illegible] “ [illegible] ” [illegible]

```
void f();
class x {void f();};
```

[illegible] x[illegible] f[illegible] f[illegible] f[illegible] x::f() [illegible] 2[illegible] _f[illegible] _x_f[illegible]

[illegible]

```
void print(char);
void print(float);
```

[illegible] _print[illegible] _print_char[illegible] _print_float[illegible]

[illegible]

5.1.1 [illegible]

[illegible]“[illegible]”[illegible]

```
void f();
int f();
```

[illegible] int x = f();[illegible] C [illegible] C++[illegible]

5.1.2 [illegible]

[illegible] C[illegible]

[illegible] C++[illegible]

□□□□□□□□□□□□□□□□□□□□□□□□□

```
//: DEF.CPP -- Function definition
  void f(int) {}
```

□□□□□□□□□□□□□□□□□□□□

```
//: USE.CPP -- Function misdeclaration
void f(char);
main() {
//! f(1);  //Causes a linker error
}
```

□□□□□□□□□□□□□□ f(int),□□□□□□□□□□□□□□□ (□□□□□□□□□) □□□□□ f(char)□□□□□□□□□□ C□□□□□□□□□□ C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ f_int□□□□□□□□□□□□□□ f_char□□□□□□□□ f_char□□□□□□ f_int□□ C++□□□□ C□□□□□□□□□□□□□□□□

5.2 □□□□□

□□ stash□□□□□□

```
//: STASH4.H -- Function overloading
#ifndef STASH4_H_
#define STASH4_H_

class stash {
  int size;  // Size of each space
  int quantity; // Number of storage spaces
  int next; // Next empty space
  // Dynamically allocated array of bytes:
  unsigned char* storage;
  void inflate(int increase);
public:
  stash(int Size); // Zero quantity
  stash(int Size, int InitQuant);
  ~stash();
  int add(void* element);
  void* fetch(int index);
  int count();
};
#endif // STASH4_H_
```

stash()□□□□□□□□□□□□□□□□□□□□□ Quantity□□□□□□□□□□□□□□□□□□□□□□□□□□□ quantity□□□□ storage□□□□□□□

```
//: STASH4.CPP -- Function overloading
#include "..\4\stash4.h"
```

```
#include <assert.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

stash::stash(int Size) {
  size = Size;
  quantity = 0;
  next = 0;
  storage = 0;
}

stash::stash(int Size, int InitQuant) {
  size = Size;
  quantity = 0;
  next = 0;
  storage = 0;
  inflate(InitQuant);
}

stash::~stash() {
  if(storage) {
    puts("freeing storage");
    free(storage);
  }
}

int stash::add(void* element) {
  if(next >= quantity) // Enough space left?
    inflate(100); // Add space for 100 elements
  // Copy element into storage,
  // starting at next empty space:
  memcpy(&(storage[next * size]),
         element, size);
  next++;
  return(next - 1); // Index number
}

void* stash::fetch(int index) {
  if(index >= next || index < 0)
    return 0;   // Not out of bounds?
  // Produce pointer to desired element:
  return &(storage[index * size]);
}

int stash::count() {
  return next; // Number of elements in stash
}

void stash::inflate(int increase) {
  void* v =
    realloc(storage, (quantity+increase)*size);
  assert(v);  // Was it successful?
  storage = (unsigned char*)v;
  quantity += increase;
}
```

[illegible] storage,[illegible] add()[illegible] add()[illegible]

[illegible]

```
//: STSHTST4.CPP -- Function overloading
#include "..\4\stash4.h"
#include <stdio.h>
#include <assert.h>
#define BUFSIZE 80

main() {
  int i;
  FILE* file;
  char buf[BUFSIZE];
  char* cp;
  // ....
  stash intStash(sizeof(int));
  for(i = 0; i < 100; i++)
    intStash.add(&i);
  file = fopen("STSHTST4.CPP", "r");
  assert(file);
  // Holds 80-character strings:
  stash stringStash(sizeof(char) * BUFSIZE);
  while(fgets(buf, BUFSIZE, file))
    stringStash.add(buf);
  fclose(file);

  for(i = 0; i < intStash.count(); i++)
    printf("intStash.fetch(%d) = %d\n", i,
           *(int*)intStash.fetch(i));
  i = 0;
  while(
    (cp = (char*)stringStash.fetch(i++)) != 0)
    printf("stringStash.fetch(%d) = %s",
           i - 1, cp);
  putchar('\n');
}
```

[illegible] stash[illegible]

5.3 [illegible]

[illegible] stash[illegible]——[illegible]

C++[illegible] stash[illegible]

```
stash(int  Size);// zero  quantity
stash(int  Size,int  Quantity);
```

[illegible]

```
stash(int Size, int  Quantity=0);
```

[illegible] stash(int)[illegible]——[illegible] stash(int,int)[illegible]

[illegible]

```
stash  A(100),B(100,0);
```

[illegible] A[illegible]

[illegible]

[illegible]

[illegible]

```
void  fn(int x  /* =0*/ ) { //...
```

[illegible]

```
void  f(int  X, int = 0, float =1.1);
```

[illegible] C++[illegible]

```
void f(int X, int,float f)  {/*...*/}
```

[illegible] x[illegible] f[illegible] ,[illegible] placeholder[illegible] ,[illegible] f(1)[illegible] f(1,2,3.0)[illegible]

[illegible]

[illegible]

[illegible]

[illegible]“ on”[illegible]“ off”[illegible]

[illegible]

```
//: FLAGS.CPP -- List of true/false flags
#include <stdio.h>
#include <string.h>
#include <assert.h>
#define FSIZE 100
#define TRUE 1
#define FALSE 0

class flags {
  unsigned char f[FSIZE];
public:
  flags();
  void set(int i);
  void clear(int i);
  int read(int i);
  int size();
};

flags::flags() {
  memset(f, FALSE, FSIZE);
}

void flags::set(int i) {
  assert(i >= 0 && i < FSIZE);
  f[i] = TRUE;
}

void flags::clear(int i) {
  assert(i >= 0 && i < FSIZE);
  f[i] = FALSE;
}

int flags::read(int i) {
  assert(i >= 0 && i < FSIZE);
  return f[i];
}

int flags::size() { return FSIZE; }

main() {
  flags fl;
  for(int i = 0; i < fl.size(); i++)
    if(i % 3 == 0) fl.set(i);
  for(int j = 0; j < fl.size(); j++)
    printf("fl.read(%d)= %d\n", j, fl.read(j));
}
```

□□□BitVector□□□□□□□□□□□□□□□□□□□□□□□bits()□□□□

```
//: BITVECT.H -- Bit Vector
#ifndef BITVECT_H_
#define BITVECT_H_

class BitVector {
  unsigned char* bytes;
  int Bits, numBytes;
public:
  BitVector(); // Default: 0 size
  // init points to an array of bytes
  // size is measured in bytes
  BitVector(unsigned char* init,
            int size = 8);
  // binary is a string of 1s and 0s
  BitVector(char* binary);
  ~BitVector();
  void set(int bit);
  void clear(int bit);
  int read(int bit);
  int bits(); // Number of bits in the vector
  void bits(int sz); // Set number of bits
  void print(const char* msg = "");
};
#endif // BITVECT_H_
```

□□□□□□□□□□□□□□□□□□□□□□□□□BitVector□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□bits()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□bits(int)□□□

——[illegible]

[illegible] 8[illegible]

[illegible] BitVector b(0)[illegible] 8[illegible] BitVector [illegible] char* [illegible] 0[illegible] BitVector[illegible] BitVector b((unsigned char *)0)[illegible] BitVector b[illegible] b.bits(64),[illegible] 8[illegible]

[illegible] char*[illegible] unsigned char*[illegible] BitVector(unsigned char*,int)[illegible] BitVector(char*)[illegible]

[illegible] print()[illegible] char*[illegible]

[illegible]

BitVector[illegible] 0110010[illegible]

[illegible] set()[illegible] clear()[illegible] read(),[illegible] print()[illegible] BitVector[illegible] 0[illegible] 1[illegible]

[illegible] BitVector[illegible] 8[illegible] BitVector[illegible] print()[illegible]

[illegible]

```
//: BITVECT.CPP -- BitVector Implementation
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
#include <string.h>
#include "..\4\bitvect.h"
#include <limits.h> //CHAR_BIT = # bits in char
// A byte with the high bit set:
const unsigned char highbit =
  1 << (CHAR_BIT - 1);

BitVector::BitVector() {
  numBytes = 0;
  Bits = 0;
  bytes = 0;
}
// Notice default args are not duplicated:
BitVector::BitVector(unsigned char* init,
                     int size) {
  numBytes = size;
  Bits = numBytes * CHAR_BIT;
  bytes = (unsigned char*)calloc(numBytes, 1);
  assert(bytes);
  if(init == 0) return; // Default to all 0
  // Translate from bytes into bit sequence:
  for(int index = 0; index<numBytes; index++)
    for(int offset = 0;
        offset < CHAR_BIT; offset++)
      if(init[index] & (highbit >> offset))
         set(index * CHAR_BIT + offset);
}

BitVector::BitVector(char* binary) {
  Bits = strlen(binary);
  numBytes =  Bits / CHAR_BIT;
  // If there's a remainder, add 1 byte:
  if(Bits % CHAR_BIT) numBytes++;
  bytes = (unsigned char*)calloc(numBytes, 1);
  assert(bytes);
  for(int i = 0; i < Bits; i++)
    if(binary[i] == '1') set(i);
}

BitVector::~BitVector() {
  free(bytes);
}

void BitVector::set(int bit) {
  assert(bit >= 0 && bit < Bits);
  int index = bit / CHAR_BIT;
  int offset = bit % CHAR_BIT;
  unsigned char mask = (1 << offset);
  bytes[index] |= mask;
}

int BitVector::read(int bit) {
  assert(bit >= 0 && bit < Bits);
  int index = bit / CHAR_BIT;
  int offset = bit % CHAR_BIT;
  unsigned char mask = (1 << offset);
  return bytes[index] & mask;
}

void BitVector::clear(int bit) {
  assert(bit >= 0 && bit < Bits);
  int index = bit / CHAR_BIT;
  int offset = bit % CHAR_BIT;
  unsigned char mask = ~(1 << offset);
  bytes[index] &= mask;
}

int BitVector::bits() { return Bits; }

void BitVector::bits(int size) {
  int oldsize = Bits;
  Bits = size;
  numBytes =  Bits / CHAR_BIT;
  // If there's a remainder, add 1 byte:
  if(Bits % CHAR_BIT) numBytes++;
  void* v = realloc(bytes, numBytes);
  assert(v);
  bytes = (unsigned char*)v;
  for(int i = oldsize; i < Bits; i++)
    clear(i); // Erase additional bits
}

void BitVector::print(const char* msg) {
  puts(msg);
  for(int i = 0; i < Bits; i++){
    if(read(i)) putchar('1');
      else putchar('0');
    // Format into byte blocks:
    if((i + 1) % CHAR_BIT == 0) putchar(' ');
  }
  putchar('\n');
}
```

[illegible] for[illegible] for[illegible] init[index]&(0x80>>offset)[illegible] 16[illegible] 0x80[illegible] 1[illegible] offset[illegible] 1[illegible] set()[illegible] BitVector [illegible] print()[illegible]

[illegible] 0[illegible] 1[illegible] BitVector[illegible]

[illegible] 8[illegible] numBytes[illegible] 8[illegible] 0[illegible]

set()[illegible] clear()[illegible] read()[illegible] assert()[illegible] Set()[illegible] read()[illegible] 1[illegible] set()[illegible]“[illegible]”[illegible] 1[illegible] read()[illegible]“[illegible]”[illegible] clear()[illegible] 1[illegible]“[illegible]”[illegible] ~[illegible]“[illegible]”[illegible]

[illegible] set()[illegible] read()[illegible] clear()[illegible] clear()[illegible]

bytes[bit/CHAR_BIT]&=~(1<<(bit % CHAR_BIT))[illegible]

[illegible]

[illegible] bits()[illegible] realloc()[illegible] bytes[illegible] realloc()[illegible]

print()[illegible] msg[illegible] C[illegible] puts()[illegible] read()[illegible] 8[illegible] BitVector[illegible] print()[illegible]

[illegible] BitVector[illegible] BitVector[illegible]

```
//: BVTEST.CPP -- Testing the BitVector class
#include "..\4\bitvect.h"

main() {
  unsigned char b[] = {
    0x0f, 0xff, 0xf0,
    0xAA, 0x78, 0x11
  };
  BitVector bv1(b, sizeof b / sizeof *b),
    bv2("10010100111100101010001010010010101");
  bv1.print("bv1 before modification");
  for(int i = 36; i < bv1.bits(); i++)
    bv1.clear(i);
  bv1.print("bv1 after modification");
  bv2.print("bv2 before modification");
  for(int j=bv2.bits()-10; j<bv2.bits(); j++)
    bv2.clear(j);
  bv2.set(30);
  bv2.print("bv2 after modification");
  bv2.bits(bv2.bits() / 2);
  bv2.print("bv2 cut in half");
```

```
    bv2.bits(bv2.bits() + 10);
    bv2.print("bv2 grown by 10");
    BitVector bv3((unsigned char*)0);
}
```

□□bv1□bv2□bv3□□□□□□□□BitVector□□□□□□□□□set()□clear()□□□□□□□read()□print()□□□□□□□□□□□□□□bv2□□□□□□□□□□□□□□□□□□BitVector□□□□□□□□□□□

□□□□□□□□□□C++□□□□□bits□bitstring□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

5.4 □□

□□BitVector□□□□□□□□□□□bits()□□□□□□□□

int bits(int sz=-1);

□□□□□□□□□□□□□□□□□□□□□-1□□□

□bits□□

□□□-1□□

int i=bv1.set(10);

□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□-1□□

□□□

□□□

□□

□□□□□□□□□□□

5.5 □□

1) □□□□ message□□□□□□□□□□ char*□□□□□□□□□□□□□□ char*□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□ print();□□□□□□□□□□□□□□□□□□□□□□□□□ char*□ □□ □□□□□

2) □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

3) □□□□□□□ STASH4.H□ STASH4.CPP□□□□□□□□□□□□□ stash□□□ □□□□□□□

4) □□ flags□□ BitVector□□□□□□□□□□□□□□□□□□□ set()□ clear()□ read()□□ index□ offset□ mask□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□

5) □□FLAGS.CPP□□□□□□□□□□□□□□□□□□□□□□□□□□ □□ □□□□□□□□□ 100□□□□□□□□□□□□□□□□□

[illegible] 6[illegible] [illegible]

[illegible] C[illegible] I/O[illegible]

[illegible] I/O[illegible] “[illegible]” ——[illegible] I/O[illegible] ——[illegible] C[illegible]

[illegible] C++[illegible] C I/O [illegible]

6.1 [illegible]

[illegible] C[illegible] C[illegible] C[illegible] close()[illegible]

```
//: FILECLAS.H -- Stdio files wrapped
#ifndef FILECLAS_H_
#define FILECLAS_H_
#include <stdio.h>

class file {
  FILE* f;
public:
  file(const char* fname, const char* mode="r");
  ~file();
  FILE* fp();
};
#endif // FILECLAS_H_
```

[illegible] C[illegible] I/O[illegible] “ r”[illegible] “[illegible]”[illegible]

[illegible] I/O[illegible] fp( )[illegible]

```
//: FILECLAS.CPP -- Stdio files wrapped
#include <stdlib.h>
#include "..\5\fileclas.h"

file::file(const char* fname, const char* mode){
```

China-pub.com [illegible]

```
  f = fopen(fname, mode);
  if(f == NULL) {
    printf("%s: file not found\n", fname);
    exit(1);
  }
}

file::~file() {
  fclose(f);
}

FILE* file::fp() {
  return f;
}
```

[illegible] fopen( )[illegible] exit( )[illegible] fp( )[illegible] f[illegible]

```
//: FCTEST.CPP -- Testing class file
#include <assert.h>
#include "..\5\fileclas.h"

main(int argc, char* argv[]) {
  assert(argc == 2);
  file f(argv[1]); // Opens and tests
  #define BSIZE 100
  char buf[BSIZE];
  while(fgets(buf, BSIZE, f.fp()))
    puts(buf);
} // File automatically closed by destructor
```

[illegible] I/O[illegible] fp( )[illegible]

[illegible]

[illegible] fp( )[illegible] fp() [illegible] f[illegible] f——[illegible] fp()[illegible]

[illegible] I/O[illegible] C[illegible] C++[illegible]

```
//: FULLWRAP.H -- Completely hidden file IO
#ifndef FULLWRAP_H_
#define FULLWRAP_H_
```

```
#include <stdio.h>

class File {
  FILE* f;
  FILE* F(); // Produces checked pointer to f
public:
File(); // Create object but don't open file
File(const char* path,
     const char* mode = "r");
~File();
int open(const char* path,
         const char* mode = "r");
int reopen(const char* path,
           const char* mode);
int Getc();
int Ungetc(int c);
int Putc(int c);
int puts(const char* s);
char* gets(char* s, int n);
int printf(const char* format, ...);
size_t read(void* ptr, size_t size,
            size_t n);
size_t write(const void* ptr,
             size_t size, size_t n);
int eof();
int close();
int flush();
int seek(long offset, int whence);
int getpos(fpos_t* pos);
int setpos(const fpos_t* pos);
long tell();
void rewind();
void setbuf(char* buf);
int setvbuf(char* buf, int type, size_t sz);
int error();
void Clearerr();
};
#endif // FULLWRAP_H_
```

□□□□□□□□□□□□ STDIO.H□□□ I/O□□□□□ vfprintf()□□□□□□□□□□ printf()□□□□□

File□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ File□□□□□□□□□ File□□□。□□□□□□□□□□□□□□

□□□□□□□□□□ File□□ f□ 0□□□□□□□ f□□□□□□□□□□□□□□□□

[illegible]0[illegible]F[illegible]F[illegible]File[illegible] [1]

[illegible]I/O[illegible]/[illegible]

[illegible]

1) [illegible]

```
printf("%c",'x');
```

[illegible]

2) [illegible]printf()[illegible]

3) [illegible]printf()[illegible]C++[illegible]I/O[illegible]

4) [illegible]C++[illegible]printf()[illegible]C[illegible]printf()[illegible]scanf()[illegible]printf()[illegible]C++[illegible]

6.2 [illegible]

[illegible]C++[illegible]I/O[illegible]“hello,world”[illegible], [illegible]I/O[illegible]C++[illegible]I/O[illegible]

[illegible]I/O[illegible]C++[illegible]

6.2.1 [illegible]

[illegible]C++[illegible]“<<”[illegible]“>>”[illegible]“[illegible]”[illegible]

[illegible]5[illegible]

---

[1] FULLWRAP test file[illegible]

[illegible]

[illegible] C++[illegible]

[illegible] C[illegible] C++[illegible]

1) [illegible] C[illegible] C++[illegible] C++[illegible] C[illegible] “[illegible]”[illegible] C++[illegible] C[illegible]

2) [illegible] C++[illegible] C ++[illegible] C[illegible]

[illegible] C[illegible] C++[illegible] ——[illegible]

```
1<<4
```

[illegible]

```
1.414<<1;
```

[illegible]

6.2.2 [illegible]

[illegible]“ <<”[illegible]“ >>”[illegible]

[illegible] istream[illegible] ostream[illegible] (ifstreams)[illegible] (ofstreams)[illegible] char* [illegible] istrstreams[illegible] (ostrstreams)[illegible] C++[illegible] string[illegible] istringstreams[illegible] (ostringstreams)[illegible] I/O[illegible]

[illegible] cin[illegible] cin[illegible] C[illegible] stdin[illegible] IOSTREAM.H[illegible]

```
int i;
cin >> i;

float f;
cin >> f;

char c;
cin >> c;

char buf[100];
cin >> buf;
```

□□□□□□□□□□□□□“ >>”□□□□□□□□□□□□□□□□□“ >>”□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□“ <<” □ cout□□ (□□□□□□□□□□□□□□□□□□□□□□□□ cerr□□)□

```
cout << "i = ";
cout << i;
cout << "\n";
cout << "f = ";
cout << f;
cout << "\n";
cout << "c = ";
cout << c;
cout << "\n";
cout << "buf = ";
cout << buf;
cout << "\n";
```

□□□□□□□□□□□□ printf()□□□

```
cout << "i = " << i << endl;
cout << "f = " << f << endl;
cout << "c = " << c << endl;
cout << "buf = " << buf << endl;
```

□□□

1. □□□□

□□□□□□□□□□□□□□□□□□ endl□□□

```
cout<<flush□
```

□□□□□□□□□□□□□□□□□ oct(□□□)□ dec(□□□)□ hex(□□□□)□

```
cout<<hex<<"0x"<<i<<endl□
```

□□□□□□□□□□□□“ □□” □□□

```
cin>>ws□
```

□□□□□ ends□□□□□□ endl□□□□□□□□□□□ strstreams□□□□□□□□□□ IOSTREAM.H□□□□□□□□□ IOMANIP.H□□□□□□□□□□□□□□□□

6.2.3 □□□□

□□ cin□□□□“ >>” □ cout□□□□“ <<” □□□ scanf()□□

□□□□□□□□□□□□□

```
//: IOSEXAMP.CPP -- Iostream examples
#include <iostream.h>

main() {
  int i;
  cin >> i;

  float f;
  cin >> f;

  char c;
  cin >> c;

  char buf[100];
  cin >> buf;

  cout << "i = " << i << endl;
  cout << "f = " << f << endl;
  cout << "c = " << c << endl;
  cout << "buf = " << buf << endl;

  cout << flush;
 cout << hex << "0x" << i << endl;
}
```

□□□□□□□

```
12 1.4 c this is a test
```

□□□□□□□□□□□□□□

```
12
1.4
c
this is a test
```

□□□□□、□□□□□□□□□□

```
i = 12
f = 1.4
c = c
buf = this
0xc
```

□□□ buf□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ “ this” □□□□□□□□□□□□□□□□□ buf□□□□□□□□□□ buf□□□□

□□□□ cin□□□

[illegible] GUI[illegible] I/O[illegible] cin[illegible]

1) [illegible]——[illegible] GUI[illegible]

2) [illegible]

3) [illegible] cout[illegible] cout[illegible] cout[illegible]

6.2.4 [illegible]

[illegible] get[illegible] getline[illegible] “ \n”[illegible]

[illegible] get[illegible] get[illegible] get[illegible] getline[illegible]

[illegible] getline[illegible]

1. get[illegible]

get[illegible] int[illegible] 10[illegible]

2. [illegible]

[illegible] read[illegible] write[illegible]

3. [illegible]

[illegible] get[illegible] get[illegible] getline[illegible] get[illegible] EOF[illegible] good[illegible] eof[illegible] fail[illegible] bad[illegible] eofbit[illegible] failbit[illegible] badbit[illegible]

[illegible] read[illegible]

[illegible] while(cin)[illegible] if(cin)[illegible] if[illegible] cin[illegible] “ [illegible] ”[illegible]

[illegible] eofbit[illegible] failbit[illegible] clear()[illegible] [1][illegible]

6.3 [illegible]

[illegible] C[illegible] STDIO.H[illegible] close[illegible] ofstream[illegible] FSTREAM.[illegible] I/O[illegible] IOSTREAM.H[illegible]

```
//: STRFILE.CPP -- Stream I/O with files
// The difference between get() & getline()
#include <fstream.h>  // Includes iostream.h
#include <assert.h>
#define SZ 100  // Buffer size

main() {
  char buf[SZ];
  {
    ifstream in("strfile.cpp"); // Read
    assert(in); // Ensure successful open
    ofstream out("strfile.out"); // Write
    assert(out);
    int i = 1; // Line counter

    // A less-convenient approach for line input:
    while(in.get(buf, SZ)) { // Leaves \n in input
      in.get(); // Throw away next character (\n)
      cout << buf << endl; // Must add \n
      // File output just like standard I/O:
      out << i++ << ": " << buf << endl;
    }
  } // Destructors close in & out

  ifstream in("strfile.out");
  assert(in);
```

---

[1] [illegible]

```
// More convenient line input:
while(in.getline(buf, SZ)) { // Removes \n
  char* cp = buf;
  while(*cp != ':')
    cp++;
  cp += 2; // Past ": "
  cout << cp << endl; // Must still add \n
  }
}
```

□□ ifstream□ ofstream□□□□□ assert()□□ (□□□□□□□□□□□□□□□□□□□□□ ,□□□□□□ 11□□□)□

□□□ while□□□□ get()□□□□□□□□□□□□□□□□□□□□□ SZ-1□□□□□□□□□□□□□□□□□□□□□□ “ \n”□□ get□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ get□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ get□ □□□□□□□□□□□□□□ in.get()□□□□□□□□ get□□□□□□□□□□□□□□□□□□□□□□ int□□□□□□□□□□□□ ignore□□□ 1□□□□□□□ ignore()□□□□□□□□□□□□□□□□□□□□□□□□ EOF□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ cout□ out□□ ostream□□□

□□□ getline□ □□ out□□□ close□ □□□□□□□□□□□□□□□□ open□□□□□□□□□□ in□□□□□□□□□□ 12□□□□□□□□□□□□□□□□□□□□□□□□

□□□ while□□□□ getline□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ “ \n”□□□□ getline()□ get()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□	□□
ios::in	□□□□□□□□□□□□□□□□□□□□□ ifstream□□□□□□□□□□□□□□□□□□□□□
ios::out	□□□□□□□□□□□□□□□□□□□□□ ios::app□ ios::ate□ ios::in□ ofstream□□ ios::trunc□□□
ios::app	□□□□□□□□□□□□□□
ios::ate	□□□□□□□□□□□□□□□□□□□□□□□
ios::nocreate	□□□□□□□□□□□□□□□□
ios::noreplace	□□□□□□□□□□□□□□□□□
ios::trunc	□□□□□□□□□□□□□□□□□□□□
ios::binary	□□□□□□□□□□□□□□□□□□

□□□□□□□□“ □□ ”□ OR□□□□□□□

6.4 □□□□□□□

□□□ I/O□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□ streambuf□□□□□□□□□□□□□□□□□□□□□□□□□ streambuf□□□□□□□□□□□□□□ I/O□□□□□□□□□□□□□□□□□□□□□ streambuf□□□□□□□ streambuf□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ streambuf□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ streambuf□□□□□□□□□□□□□□□□ streambuf□□□□□□□□□□□□□□

□□□□□□□□ streambuf□□□□□□□□□□□□ rdbuf()□□□□□□□□□□□□□□□□□□□□ streambuf□□□□□□□□□□□□□□ streambuf□□□□□□□□□□□□ streambuf□□□□□□□□□□□□□□□□“ <<”□□“ <<”□□

□□

```
//: STYPE.CPP -- Type a file to standard output
#include <fstream.h>
#include <assert.h>

main(int argc, char* argv[]) {
  assert(argc == 2); // Must have a command line
  ifstream in(argv[1]);
  assert(in); // Exits if it doesn't exist
  cout << in.rdbuf(); // Outputs entire file
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ ifstream□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ assert(in)□□□

□□□□□□□□□□□□□□□□□

cout<<in.rdbuf()□

□□□□□□□□□□□ cout□□□□□□□□□□□□□□□□□□□□□□□□□

□□□ streambuf□ get()□□

□□□ get□□□□□□□□□□□□□ streambuf□□□□□□□□ streambuf□□□

China-pub.com [illegible]

[illegible] 10[illegible] get[illegible]

```
//: SBUFGET.CPP -- Get directly into a streambuf
#include <fstream.h>

main() {
  ifstream in("sbufget.cpp");
  while(in.get(*cout.rdbuf()))
    in.ignore();
}
```

rdbuf()[illegible] get[illegible] ignore[illegible] get()[illegible]

[illegible]

6.5 [illegible]

[illegible]“[illegible]”[illegible] streampos[illegible] C[illegible] fseek()[illegible]

[illegible] streampos[illegible]“ tell”[illegible] tellp[illegible] tellg[illegible]“ p”[illegible]“ [illegible]”[illegible]“ g”[illegible]“ [illegible]”[illegible] streampos[illegible] seekp[illegible] seekg[illegible] streampos[illegible]

[illegible] seekp[illegible] seekg[illegible]

Ios::beg	[illegible]
Ios::cur	[illegible]
Ios::end	[illegible]

[illegible] C[illegible] C++[illegible] STDIO.H[illegible] C++[illegible] ([illegible] cin[illegible] cout[illegible])[illegible]

```
//: SEEKING.CPP -- Seeking in iostreams
#include <fstream.h>
#include <assert.h>

main(int argc, char* argv[]) {
  assert(argc == 2);
  ifstream in(argv[1]);
  assert(in); // File must already exist
  in.seekg(0, ios::end); // End of file
  streampos sp = in.tellg(); // Size of file
cout << "file size = " << sp << endl;
in.seekg(-sp/10, ios::end);
streampos sp2 = in.tellg();
in.seekg(0, ios::beg); // Start of file
cout << in.rdbuf(); // Print whole file
  in.seekg(sp2); // Move to streampos
  // Prints the last 1/10th of the file:
  cout << endl << endl << in.rdbuf() << endl;
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□ ifstream□□□□ assert□□□□□□□□□□□□□□□□□□□□□□□□□□□□ seekg□□□□□“□□□”□□□□□□□□□□□□□□□□□□□□□□□□□□□ streampos□□□ long□ typedef□□□□□ tellg□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 1/10□ ——□□ streampos□□□ sp2□□□□ seekg□□□□□□□□□□□□□□□□□□□□ rdbuf□□□□□ streambuf□□□□□□□□□□ seekg□□□□□□ streampos sp2□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□/□□□

□□□□ streambuf□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ ifstream□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ ifstream□□□□□□□□□□□□□□□□□□□□□□ ostream□□

```
ifstream in("filename", ios::in|ios::out);
ostream out(in.rdbuf());
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
//: IOFILE.CPP -- Reading & writing one file
#include <fstream.h>
main() {
  ifstream in("iofile.cpp");
  ofstream out("iofile.out");
  out << in.rdbuf(); // Copy file
  in.close();
  out.close();
  // Open for reading and writing:
  ifstream in2("iofile.out",ios::in|ios::out);
  ostream out2(in2.rdbuf());
  cout << in2.rdbuf();  // Print whole file
  out2 << "Where does this end up?";
  out2.seekp(0, ios::beg);
  out2 << "And what about this?";
  in2.seekg(0, ios::beg);
  cout << in2.rdbuf();
}
```

□□□□□□□□□□□□□□□□□□□ iofile.out□□□□□□□□□□□□□□□□ □□ □□□□□□□□□□□□□□ cout<<in2.rdbuf()□□□□□“□”□□□□□□□□□□□ □□“□”□□□□□□□□□□□□□□□“ Where does this end up?”□□□□□□□□□ □□□□□□“□”□□□□ seekp□□□□□□□□□□□□□□□□□□□□□□□“□” □□□ seekg□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ out2 □□□□□□□□□□□□□□□□□□□□□□□□□□□

6.6 strstreams

□□ □□ □□□□□□□□□□

strstream□□□□□□□□□□□□□□□□□□□□□□□□□□□ strstream□□□□ □□□□ istrstream□□□□□□□□□□ strstream□□□□□□□□ ostrstream□

□□ □□□□□□□□□□□□□□□□□□□□□□□□□□ strstreams□□□□□□□□□□□ □□□□□□

6.6.1 □□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ istrstreams□□□□ □□□□□□□□□□□□□□□□□

```
istrstream::istrstream(char* buf);
istrstream::istrstream(char* buf, int size);
```

□□ □□□ buf[size]□□□□□□□□□□□□

□□□□□□□□□□ istrstream□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□[1]□

```
//: ISTRING.CPP -- Input strstreams
#include <strstrea.h>

main() {
  istrstream s("1.414 47 This is a test");
  int i;
  float f;
  s >> i >> f; // Whitespace-delimited input
  char buf2[100];
  s >> buf2;
  cout << "i = " << i << ", f = " << f;
```

---

[1] □□□□□□□□□□□□□□ DOS□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□

```
  cout << " buf2 = " << buf2 << endl;
  cout << s.rdbuf(); // Get the rest...
}
```

□□□□C□□atof()□atoi()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□

istrstream s("1.414 47 This is a test")□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□s>>i>>f□□□□□□□□□□□i□□□□□□□□□f□□□□“□□□□□□□□□□□□”□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“1.414 47 This is a test”□□□i□□1□□□□□□□□□□□□□□□□□□□□□□□f□0.414□□

□□□□□□□□□□buf2□□rdbuf□□□

□□strstreams□□

ostrstream::ostrstream(char*,int,int=ios::out)□

□□ios::ate□ios::app□□

□□□

□□ostrstreams□□ends□

□□□□□□□□ostrstream□□

```
//: OSTRING.CPP -- Output strstreams
#include <strstrea.h>
#define SZ 100

main() {
cout << "type an int, a float and a string:";
int i;
float f;
cin >> i >> f;
cin >> ws; // Throw away white space
char buf[SZ];
cin.getline(buf, SZ); // Get rest of the line
```

□□

```
    // (cin.rdbuf() would be awkward)
    ostrstream os(buf, SZ, ios::app);
    os << endl;
    os << "integer = " << i << endl;
    os << "float = " << f << endl;
    os << ends;
    cout << buf;
    cout << os.rdbuf(); // Same effect
    cout << os.rdbuf(); // NOT the same effect
}
```

□□□□□□ int□ float□□□□□□□□□□□□□□□□□□□□□□□□□ rdbuf (□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ control-Z(□ unix□ control-D)□□□□□□□□□□□□□□ getline□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□ buf□ buf□□□□ ostrstream os □□□□□ □□□□□□ ios::app□□□□□□□□□□ buf□□□□□□□□□□□□□□□“□□”□ □□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□ ostrstream□□□□□□ ends□ □□□□□□□□□ endl□□□□□□□□□□□□□□□□□□

□□□□□ buf□□□□□□□ cout<<buf□□□□□□□□□□□□□□ os.rdbuf()□□□□ □□□□□□□□□□□ streambuf□□“□”□□□□□□□□□□□□□□□□□□□□□ □□□□ cout<<os.rdbuf()□□□□□□□□ ——“□”□□□□□□□□

6.6.2 □□□□□□

□□ strstreams□□□□ istrstreams□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□ ostrstream□

ostrstream A□

□□□ A□□□□□□□□□□□□□□□□□□□ A□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ strstream□□□□□□ streambuf□□□□□□□□□□□□□□□□□

```
A << "hello, world. i = " << i << endl << ends;
cout << A.rdbuf();
```

□□□□□□□□□□□□□□□□□□□ A□□□□□□□□□□□□□□□□□ □□□□□□□□□□□ ——□□□□ str□□□□□□□□

char* cp=A.str()□

□□□□□□□□□□□ A□□□□□□□□□□□□□□□□□□□□ A□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ A□□□□□□□□ □□□□□□□□ A□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□ □□□□□□

ostrstream□□□□□□□□□□“□□”□□□□□□□□ str□□□□□ char*□□

□□
□□□□□□□□□□

□□□□□□□ str□ □□ ostrstream□ “ □□ ” □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ ostrstream□ □□□□□□□□□□□□□□□□ ostrstream□ □□□□□□□□□□□□□ str()□ □ char*□ □□□□□□□□□□

□□ C++□ □□□□□□□□ new□ delete□ □□□ 12□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ C□ malloc()□ free()□ □□□□ new□ □□□□□□□□ delete□ □□□□□□□□□□□□□□□□□□□ C++□ □□□□□□□□ new□ □□□ ostrstream□ □□□□□□□□□□□ new□ □□□□□□□ delete□ □□□□□□□□□□ ostrstream A□ □ str()□ □ char*□ □□□□□□□□□□

delete A.str()□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ ostrstream□ □□□□□ freeze()□ □□ freeze()□ ostrstream□ streambuf□ □□□□ freeze□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□

A.rdbuf()->freeze(0)□

□ A□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ A□ □□□□□□□□□□□□□□□□□□□□□□□□□ str□ □□□□□□ ——□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□

```
//: WALRUS.CPP -- Freezing a strstream
#include <strstrea.h>

main() {
  ostrstream s;
  s << "'The time has come', the walrus said,";
  s << ends;
  cout << s.str() << endl; // String is frozen
  // S is frozen; destructor won't delete
  // the streambuf storage on the heap
  s.seekp(-1, ios::cur); // Back up before NULL
  s.rdbuf()->freeze(0); // Unfreeze it
  // Now destructor releases memory, and
  // you can add more characters (but you
  // better not use the previous str() value)
  s << " 'To speak of many things'" << ends;
  cout << s.rdbuf();
}
```

□□□□□□□ s□ □□□□□□ ends□ □□□□□□□□ str()□ □□ char*□ □□□□□□□□□□□ s□ □□□□□□□□□□□□□ s□“ □ ” □□□□□□□□□□□□□□□□□□□□ ends□ □□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□ □□□□□ seekp□ □□□□□□□□□□□ rdbuf()□ □□ freeze(0)□ □□□ streambuf□ □□ s□ □□□□□□□□□□□

s□□□□□□□ str()□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□ ostrstream□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□ ostrstream□ char*□□□□□□□□□□□□□□□□□□ ostrstream□□□□□ rdbuf() □□□□□□□□□□□□□□□□□□□□□□□□□□ ostrstream□□

1. □□□□

□□□□□□□□□□ str()□□□ ostrstream□□□□□□□□□□□□□□□□□□□□ □□□□□□□ str□□□□□□□□□□□□

```
//: STRMOVE.CPP -- Ostrstream memory movement
#include <strstrea.h>

main() {
  ostrstream s;
  s << "hi";
  char* old = s.str(); // Freezes s
  s.rdbuf()->freeze(0); // Unfreeze
  for(int i = 0; i < 100; i++)
    s << "howdy"; // Should force reallocation
  cout << "old = " << (void*)old << endl;
  cout << "new = " << (void*)s.str(); // Freezes
  delete s.str(); // Release storage
}
```

□□□□□□□ s□□□ str□□□□ char*□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□ char*□□□□□□□□ delete□□□□□□□□□□ str()□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□ char*□□□ void*□ char*□□□□ “ <<”□□□□□□□□□□□□□□ void*□□□□ “ <<”□□□□□□□□□□□□□□

□□□□□□□□□□□□ str()□□□□□□□□ s□□□□□ 0□□□□□□□□ □□□□□□ ostrstream□□□□□□□□□□

2. □□□□□□□

□□ C++ string□□□□□□□□□□□□ stringstream□ □□□□□□□□□□□□□□□□ □□□□□□□□ char*□ strstream□□□□□□□□□□□□□□ ——□□□□□□□□ [1]□

6.7 □□□□□□

□□ □□□□□□□□□□□□□□□□□ printf()□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□ □□□ □□□ □□□

---

[1] □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

[illegible]

6.7.1 [illegible]

ios[illegible] IOSTREAM.H[illegible] ios::flags()[illegible] long(typedefed to fmtflags)[illegible]

```
fmtflags ios::flags(fmtflags newflags);
fmtflags ios::setf(fmtflags ored_flag);
fmtflags ios::unsetf(fmtflags clear_flag);
fmtflags ios::setf(fmtflags bits, fmtflags field);
```

[illegible]

setf()[illegible] on[illegible] off[illegible] on/off[illegible] setf(fmtflags)[illegible] on[illegible] unsetf(fmtflags)[illegible] off[illegible]

on/off[illegible]	[illegible]
ios::skipws	[illegible]
ios::showbase	[illegible] C++[illegible]
ios::showpoint	[illegible]
ios::uppercase	[illegible] A-F[illegible] E
ios::showpos	[illegible] +[illegible]
ios::unitbuf	“[illegible]”[illegible]
ios::stdio	[illegible] C[illegible] I/O[illegible]

[illegible] cout[illegible] cout.setf(ios::showpos)[illegible] cout.unsetf(ios::showpos)[illegible]

[illegible] C[illegible] I/O[illegible] C[illegible] ios::stdio[illegible] printf()[illegible]

1. [illegible]

[illegible] ——[illegible] setf[illegible] ios::basefield[illegible] ios[illegible] dec[illegible] setf[illegible] ios::hex[illegible] ios::hex[illegible] ios::dec[illegible] setf()[illegible] setf(ios::hex,ios::basefield)[illegible] ios::basefield[illegible] ios::hex[illegible] setf()[illegible] “[illegible]”[illegible] hex()[illegible]

[illegible] set()[illegible]
[illegible]

iso::basefield	[illegible]
ios::dec	[illegible]
ios::hex	[illegible]
ios::oct	[illegible]

ios::floatfield	[illegible]
ios::scientific	[illegible]
ios::fixed	[illegible]
“ automatic” (Neither bit is set)	[illegible]

ios::adjustfield	[illegible]
ios::left	[illegible]
ios::right	[illegible]
ios::internal	[illegible]

2. [illegible]

[illegible]

function	[illegible]
int ios::width()	[illegible] 0[illegible]
int ios::width(int n)	[illegible]
int ios::fill()	[illegible]
int ios::fill(int n)	[illegible]
int ios::precision()	[illegible] 6[illegible]
int ios::precision(int n)	[illegible] “ [illegible] ” [illegible] ios::floatfield[illegible]

[illegible] 0[illegible] 123 [illegible] 2[illegible] 123[illegible]

[illegible] 0[illegible] width()[illegible]

6.7.2 [illegible]

[illegible]

```
//: FORMAT.CPP -- Formatting functions
#include <fstream.h>
#define D(a) T << #a << endl; a
ofstream T("format.out");

main() {
```

```
D(int i = 47;)
D(float f = 2300114.414159;)
char* s = "Is there any more?";

D(T.setf(ios::unitbuf);)
D(T.setf(ios::stdio);)

D(T.setf(ios::showbase);)
D(T.setf(ios::uppercase);)
D(T.setf(ios::showpos);)
D(T << i << endl;) // Default to dec
D(T.setf(ios::hex, ios::basefield);)
D(T << i << endl;)
D(T.unsetf(ios::uppercase);)
D(T.setf(ios::oct, ios::basefield);)
D(T << i << endl;)
D(T.unsetf(ios::showbase);)
D(T.setf(ios::dec, ios::basefield);)
D(T.setf(ios::left, ios::adjustfield);)
D(T.fill('0');)
D(T << "fill char: " << T.fill() << endl;)
D(T.width(10);)
T << i << endl;
D(T.setf(ios::right, ios::adjustfield);)
D(T.width(10);)
T << i << endl;
D(T.setf(ios::internal, ios::adjustfield);)
D(T.width(10);)
T << i << endl;
D(T << i << endl;) // Without width(10)

D(T.unsetf(ios::showpos);)
D(T.setf(ios::showpoint);)
D(T << "prec = " << T.precision() << endl;)
D(T.setf(ios::scientific, ios::floatfield);)
D(T << endl << f << endl;)
D(T.setf(ios::fixed, ios::floatfield);)
D(T << f << endl;)
D(T.setf(0, ios::floatfield);) // Automatic
D(T << f << endl;)
D(T.precision(20);)
D(T << "prec = " << T.precision() << endl;)
D(T << endl << f << endl;)
D(T.setf(ios::scientific, ios::floatfield);)
D(T << endl << f << endl;)
D(T.setf(ios::fixed, ios::floatfield);)
```

```
    D(T << f << endl;)
    D(T.setf(0, ios::floatfield);) // Automatic
    D(T << f << endl;)

    D(T.width(10);)
    T << s << endl;
    D(T.width(40);)
    T << s << endl;
    D(T.setf(ios::left, ios::adjustfield);)
    D(T.width(40);)
    T << s << endl;

    D(T.unsetf(ios::showpoint);)
    D(T.unsetf(ios::unitbuf);)
    D(T.unsetf(ios::stdio);)
}
```

[illegible] D[illegible] a[illegible] “stringizing” [illegible] a[illegible] D[illegible] a[illegible] a[illegible] a[illegible] T[illegible]

```
int i = 47;
float f = 2300114.414159;
T.setf(ios::unitbuf);
T.setf(ios::stdio);
T.setf(ios::showbase);
T.setf(ios::uppercase);
T.setf(ios::showpos);
T << i << endl;
+47
T.setf(ios::hex, ios::basefield);
T << i << endl;
+0X2F
T.unsetf(ios::uppercase);
T.setf(ios::oct, ios::basefield);
T << i << endl;
+057
T.unsetf(ios::showbase);
T.setf(ios::dec, ios::basefield);
T.setf(ios::left, ios::adjustfield);
T.fill('0');
T << "fill char: " << T.fill() << endl;
fill char: 0
T.width(10);
+470000000
T.setf(ios::right, ios::adjustfield);
T.width(10);
```

```
0000000+47
T.setf(ios::internal, ios::adjustfield);
T.width(10);
+000000047
T << i << endl;
+47
T.unsetf(ios::showpos);
T.setf(ios::showpoint);
T << "prec = " << T.precision() << endl;
prec = 6
T.setf(ios::scientific, ios::floatfield);
T << endl << f << endl;

2.300115e+06
T.setf(ios::fixed, ios::floatfield);
T << f << endl;
2300114.500000
T.setf(0, ios::floatfield);
T << f << endl;
2.300115e+06
T.precision(20);
T << "prec = " << T.precision() << endl;
prec = 20
T << endl << f << endl;

2300114.50000000020000000000
T.setf(ios::scientific, ios::floatfield);
T << endl << f << endl;

2.30011450000000020000e+06
T.setf(ios::fixed, ios::floatfield);
T << f << endl;
2300114.50000000020000000000
T.setf(0, ios::floatfield);
T << f << endl;
2300114.50000000020000000000
T.width(10);
Is there any more?
T.width(40);
0000000000000000000000Is there any more?
T.setf(ios::left, ios::adjustfield);
T.width(40);
Is there any more?0000000000000000000000
T.unsetf(ios::showpoint);
T.unsetf(ios::unitbuf);
T.unsetf(ios::stdio);
```

[illegible]

6.8 [illegible]

[illegible] C++ [illegible]

[illegible] IOSTREAM.H[illegible] dec[illegible] oct[illegible] hex[illegible] setf(ios::dec,ios::basefield)[illegible] setf(ios::oct,ios::basefield)[illegible] setf(ios::hex,ios::basefield)[illegible] IOSTREAM.H[1] [illegible] ws[illegible] endl[illegible] ends[illegible] flush[illegible]

[illegible]	[illegible]
showbase	[illegible]
noshowbase	[illegible] C++[illegible]
showpos	[illegible] +[illegible]
noshowpos	
uppercase	[illegible] A-F[illegible] E
nouppercase	
showpoint	[illegible]
noshowpoint	
skipws	[illegible]
noshipws	
left	[illegible]
right	[illegible]
internal	[illegible]
scientific	[illegible]
fixed	setprecision()[illegible] ios::precision()[illegible]

[illegible]

[illegible] IOMANIP.H[illegible]

[illegible]	[illegible]
setiosflags(fmtflags n)	[illegible] n[illegible] ios::setf()[illegible]
resetiosflags(fmtflags n)	[illegible] n[illegible] ios::unsetf()[illegible]
setbase(base n)	[illegible] n[illegible] n[illegible] 10[illegible] 8[illegible] 16[illegible] 0[illegible] n[illegible] 0[illegible] 10[illegible] C[illegible] 10[illegible] 10[illegible] 010 [illegible] 8[illegible] 0xf[illegible] 15[illegible] dec[illegible] oct[illegible] hex[illegible]
setfill(char n)	[illegible] n[illegible] ios[illegible] fill[illegible]
setprecision(int n)	[illegible] n[illegible] ios[illegible] precision[illegible]
setw(int n)	[illegible] n[illegible] ios[illegible] width[illegible]

[illegible]

---

[1] [illegible]

□□□□□□□□□□□□□□□□□□□□□□□□□□

```
//: MANIPS.CPP -- FORMAT.CPP using manipulators
#include <fstream.h>
#include <iomanip.h>

main() {
  ofstream T("trace.out");
  int i = 47;
  float f = 2300114.414159;
  char* s = "Is there any more?";

  T << setiosflags(
         ios::unitbuf | ios::stdio
         | ios::showbase | ios::uppercase
         | ios::showpos
       );
  T << i << endl; // Default to dec
  T << hex << i << endl;
  T << resetiosflags(ios::uppercase)
    << oct << i << endl;
  T.setf(ios::left, ios::adjustfield);
  T << resetiosflags(ios::showbase)
    << dec << setfill('0');
  T << "fill char: " << T.fill() << endl;
  T << setw(10) << i << endl;
  T.setf(ios::right, ios::adjustfield);
  T << setw(10) << i << endl;
  T.setf(ios::internal, ios::adjustfield);
  T << setw(10) << i << endl;
  T << i << endl; // Without setw(10)

  T << resetiosflags(ios::showpos)
    << setiosflags(ios::showpoint)
    << "prec = " << T.precision() << endl;
  T.setf(ios::scientific, ios::floatfield);
  T << f << endl;
  T.setf(ios::fixed, ios::floatfield);
  T << f << endl;
  T.setf(0, ios::floatfield); // Automatic
  T << f << endl;
  T << setprecision(20);
  T << "prec = " << T.precision() << endl;
  T << f << endl;
  T.setf(ios::scientific, ios::floatfield);
  T << f << endl;
  T.setf(ios::fixed, ios::floatfield);
  T << f << endl;
  T.setf(0, ios::floatfield); // Automatic
  T << f << endl;

  T << setw(10) << s << endl;
  T << setw(40) << s << endl;
  T.setf(ios::left, ios::adjustfield);
  T << setw(40) << s << endl;

  T << resetiosflags(
         ios::showpoint | ios::unitbuf
         | ios::stdio
      );
}
```

□ setiosflags()□ resetiosflags()□ □ □ □ □ □ □ □ □ □ □ □ □ “ □ OR” □ setf()□ unsetf()□ □ □ □

6.9 □ □ □ □ □ □

□ endl□ ostream□ □ □ □ □ □ □ □ □ □□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ 10□ □ □ □ □ □ endl□ □ □ □ □

ostream& endl(ostream&)□

□ □ □ □ □ □ □ □

cout<<"howdy" <<endl□

endl□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □“ □ ” □ □ □ □ □ □ □ □ □ □ □ □ □ IOSTREAM.H□ □ □ □ □ □ □ □ □ □ □ □ □ “ □ □ □ □ ”□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ostream□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

□ nl□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

```
//: NL.CPP -- Creating a manipulator
#include <iostream.h>

ostream& nl(ostream& os) {
  return os << '\n';
}

main() {
  cout << "newlines" << nl << "between" << nl
       << "each" << nl << "word" << nl;
}
```

[illegible]

os<<'\n'[illegible]

[illegible] os[illegible] nl[illegible] [1][illegible]

[illegible] nl[illegible] endl[illegible]

[illegible]

[illegible] Jerry Schwarz[illegible] [2][illegible] “ <<” [illegible] “ <<” [illegible] 11[illegible]

```
//: EFFECTOR.CPP -- Jerry Schwarz's "effectors"
#include<iostream.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <limits.h> // ULONG_MAX

// Put out a portion of a string:
class fixw {
  char* s;
public:
  fixw(const char* S, int width);
  ~fixw();
  friend ostream& operator<<(ostream&, fixw&);
};

fixw::fixw(const char* S, int width) {
  s = (char*)malloc(width + 1);
  assert(s);
  strncpy(s, S, width);
  s[width] = 0; // Null-terminate
}

fixw::~fixw() { free(s); }

ostream& operator<<(ostream& os, fixw& fw) {
  return os << fw.s;
}
```

---

[1] [illegible] nl[illegible] 8[illegible]

[2] [illegible]

```
// Print a number in binary:
typedef unsigned long ulong;

class bin {
  ulong n;
public:
  bin(ulong N);
  friend ostream& operator<<(ostream&, bin&);
};

bin::bin(ulong N) { n = N; }

ostream& operator<<(ostream& os, bin& b) {
  ulong bit = ~(ULONG_MAX >> 1); // Top bit set
  while(bit) {
    os << (b.n & bit ? '1' : '0');
    bit >>= 1;
  }
  return os;
}
main() {
  char* string =
    "Things that make us happy, make us wise";
  for(int i = 1; i <= strlen(string); i++)
    cout << fixw(string, i) << endl;
  ulong x = 0xFEDCBA98UL;
  ulong y = 0x76543210UL;
  cout << "x in binary: " << bin(x) << endl;
  cout << "y in binary: " << bin(y) << endl;
}
```

fixw char* “ <<” fixw ostream ostream fixw

cout <<fixw(string,i)<<endl

 fixw “ <<”

bin ULONG_MAX unsigned long LIMITS.H

 char* fixw unsign long bin fixw bin
9

6.10 [illegible]

[illegible] unix[illegible] sed[illegible] awk[illegible] sed[illegible] awk[illegible]

6.10.1 [illegible]

[illegible] main[illegible]

```
//: MAKEMAIN.CPP -- Create a shell main() File
#include <fstream.h>
#include <strstrea.h>
#include <assert.h>
#include <string.h>
#include <ctype.h>

main(int argc, char* argv[]) {
  assert(argc == 2);
  // Don't replace it if it exists:
  ofstream mainfile(argv[1], ios::noreplace);
  assert(mainfile);
  istrstream name(argv[1]);
  ostrstream CAPname;
  char c;
  while(name.get(c))
    CAPname << char(toupper(c));
  CAPname << ends;
  mainfile << "//:"  << ' ' << CAPname.rdbuf()
    << " -- " << endl
    << "#include <iostream.h>" << endl
    << endl
    << "main() {" << endl << endl
    << "}" << endl;
}
```

[illegible] ios::noreplace[illegible] istrstream[illegible] C[illegible] toupper()[illegible] int[illegible] char[illegible]

1. [illegible]

[illegible]

□□□□□□□□□□□□□NAME.H□□□□□NAME.CPP□□□——□□□——□□□

□□□□□□□□□□□——□□

```
//: CPPCHECK.CPP -- Configures .H & .CPP files
// To conform to style standard.
// Tests existing files for conformance
#include <fstream.h>
#include <strstrea.h>
#include <string.h>
#include <ctype.h>
#include <assert.h>
#define SZ 40  // Buffer sizes
#define BSZ 100

main(int argc, char* argv[]) {
  assert(argc == 2); // File set name
  enum bufs { base, header, implement,
    Hline1, guard1, guard2, guard3,
    CPPline1, include, bufnum };
  char b[bufnum][SZ];
  ostrstream osarray[] = {
    ostrstream(b[base], SZ),
    ostrstream(b[header], SZ),
    ostrstream(b[implement], SZ),
    ostrstream(b[Hline1], SZ),
    ostrstream(b[guard1], SZ),
    ostrstream(b[guard2], SZ),
    ostrstream(b[guard3], SZ),
    ostrstream(b[CPPline1], SZ),
    ostrstream(b[include], SZ),
  };
  osarray[base] << argv[1] << ends;
  // Find any '.' in the string using the
  // Standard C library function strchr():
  char* period = strchr(b[base], '.');
  if(period) *period = 0; // Strip extension
  // Force to upper case:
  for(int i = 0; b[base][i]; i++)
    b[base][i] = toupper(b[base][i]);
  // Create file names and internal lines:
```

```
osarray[header] << b[base] << ".H" << ends;
osarray[implement] << b[base] << ".CPP" << ends;
osarray[Hline1] << "//:" << ' ' << b[header]
  << " -- " << ends;
osarray[guard1] << "#ifndef " << b[base]
                << "_H_" << ends;
osarray[guard2] << "#define " << b[base]
                << "_H_" << ends;
osarray[guard3] << "#endif // " << b[base]
                << "_H_" << ends;
osarray[CPPline1] << "//:" << ' '
                  << b[implement]
                  << " -- " << ends;
osarray[include] << "#include \""
                 << b[header] << "\"" <<ends;
// First, try to open existing files:
ifstream existh(b[header]),
         existcpp(b[implement]);
if(!existh) { // Doesn't exist; create it
  ofstream newheader(b[header]);
  newheader << b[Hline1] << endl
    << b[guard1] << endl
    << b[guard2] << endl << endl
    << b[guard3] << endl;
}
if(!existcpp) { // Create cpp file
  ofstream newcpp(implement);
  newcpp << b[CPPline1] << endl
    << b[include] << endl;
}
if(existh) { // Already exists; verify it
  strstream hfile; // Write & read
  ostrstream newheader; // Write
  hfile << existh.rdbuf() << ends;
  // Check that first line conforms:
  char buf[BSZ];
  if(hfile.getline(buf, BSZ)) {
    if(!strstr(buf, "//:") ||
      !strstr(buf, b[header]))
    newheader << b[Hline1] << endl;
}
// Ensure guard lines are in header:
if(!strstr(hfile.str(), b[guard1]) ||
   !strstr(hfile.str(), b[guard2]) ||
   !strstr(hfile.str(), b[guard3])) {
      newheader << b[guard1] << endl
        << b[guard2] << endl
        << buf
        << hfile.rdbuf() << endl
        << b[guard3] << endl << ends;
  } else
    newheader << buf
      << hfile.rdbuf() << ends;
  // If there were changes, overwrite file:
  if(strcmp(hfile.str(),newheader.str())!=0){
    existh.close();
    ofstream newH(b[header]);
    newH << "//@//" << endl // change marker
      << newheader.rdbuf();
  }
  delete hfile.str();
  delete newheader.str();
}
if(existcpp) { // Already exists; verify it
  strstream cppfile;
  ostrstream newcpp;
  cppfile << existcpp.rdbuf() << ends;
  char buf[BSZ];
  // Check that first line conforms:
  if(cppfile.getline(buf, BSZ))
    if(!strstr(buf, "//:") ||
        !strstr(buf, b[implement]))
      newcpp << b[CPPline1] << endl;
  // Ensure header is included:
  if(!strstr(cppfile.str(), b[include]))
      newcpp << b[include] << endl;
    // Put in the rest of the file:
    newcpp << buf << endl; // First line read
    newcpp << cppfile.rdbuf() << ends;
    // If there were changes, overwrite file:
    if(strcmp(cppfile.str(),newcpp.str())!=0){
      existcpp.close();
      ofstream newCPP(b[implement]);
      newCPP << "//@//" << endl // change marker
        << newcpp.rdbuf();
    }
    delete cppfile.str();
    delete newcpp.str();
  }
}
```

[illegible] ostrstream[illegible] enum bufs[illegible] ostrstream[illegible] char[illegible] b[illegible] char[illegible] bufnum[illegible] bufnum[illegible] bufs[illegible] enum[illegible] bufnum[illegible] buf[illegible] b[illegible] SZ[illegible]

[illegible] base[illegible] header[illegible] implement[illegible] CPP[illegible] Hline1[illegible] guard1 [illegible] guard2[illegible] guard3[illegible] “[illegible]” [illegible] CPPlinel[illegible] CPP[illegible] include[illegible] CPP[illegible]

osarray[illegible] ostrstream[illegible] ostrstream[illegible] bufs[illegible] b[illegible] osarray[illegible] b[illegible] ostrstream[illegible]

[illegible] CPP[illegible] ifstreams[illegible] “[illegible]” [illegible]

[illegible] strstream[illegible] “ //[illegible]” [illegible] C[illegible] strstr()[illegible] ostrstream[illegible]

[illegible] strstr()[illegible] “[illegible]” [illegible]

[illegible] strstream[illegible] ostrstream[illegible] ofstream[illegible] ostrstream[illegible]

2. [illegible]

[illegible] “ //[illegible]” [illegible]

```
//: SHOWERR.CPP -- Un-comment error generators
#include <fstream.h>
#include <strstrea.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <assert.h>
```

```
char* marker = "//!";

char* usage =
"usage: showerr filename chapnum\n"
"where filename is a C++ source file\n"
"and chapnum is the chapter name it's in.\n"
"Finds lines commented with //! and removes\n"
"comment, appending //(#) where # is unique\n"
"across all files, so you can determine\n"
"if your compiler finds the error.\n"
"showerr /r\n"
"resets the unique counter.";

// File containing error number counter:
char* errnum = "..\\errnum.txt";
// File containing error lines:
char* errfile = "..\\errlines.txt";
ofstream errlines(errfile,ios::app);

main(int argc, char* argv[]) {
  if(argc < 2) {
    cerr << usage << endl;
    return 1;
  }
  if(argv[1][0] == '/' || argv[1][0] == '-') {
    // Allow for other switches:
    switch(argv[1][1]) {
      case 'r': case 'R':
        cout << "reset counter" << endl;
        remove(errnum); // Delete files
        remove(errfile);
        return 0;
      default:
        cerr << usage << endl;
        return 1;
    }
  }
  char* chapter = argv[2];
  strstream edited; // Edited file
  int counter = 0;
  {
    ifstream infile(argv[1]);
    assert(infile);
    ifstream count(errnum);
    if(count) count >> counter;
    int linecount = 0;
```

```
      #define sz 255
      char buf[sz];
      while(infile.getline(buf, sz)) {
        linecount++;
        // Eat white space:
        int i = 0;
        while(isspace(buf[i]))
        i++;
      // Find marker at start of line:
      if(strstr(&buf[i], marker) == &buf[i]) {
        // Erase marker:
        memset(&buf[i], ' ', strlen(marker));
        // Append counter & error info:
        ostrstream out(buf, sz, ios::ate);
        out << "//(" << ++counter << ") "
            << "Chapter " << chapter
            << " File: " << argv[1]
            << " Line " << linecount << endl
            << ends;
          edited << buf;
        errlines << buf; // Append error file
      } else
        edited << buf << "\n"; // Just copy
    }
  } // Closes files
  ofstream outfile(argv[1]); // Overwrites
  outfile << edited.rdbuf();
  ofstream count(errnum); // Overwrites
  count << counter; // Save new counter
}
```

[illegible] maker[illegible]

[illegible] maker[illegible] strstream edited[illegible] edited[illegible]

6.10.2 [illegible]

[illegible] ——[illegible]

```
//: DATALOG.H -- Datalogger record layout
#ifndef DATALOG_H_
#define DATALOG_H_
#include <time.h>
#include <iostream.h>
```

```
#define BSZ 10

class datapoint {
  tm Tm; // Time & day
  // Ascii degrees (*) minutes (') seconds ("):
  char Latitude[BSZ], Longitude[BSZ];
  double Depth, Temperature;
public:
  tm Time(); // read the time
  void Time(tm T); // set the time
  const char* latitude(); // read
  void latitude(const char* l); // set
  const char* longitude(); // read
  void longitude(const char* l); // set
  double depth(); // read
  void depth(double d); // set
  double temperature(); // read
  void temperature(double t); // set
  void print(ostream& os);
};
#endif // DATALOG_H_
```

□□□□□□□□□□□□□□□□□□□□□□□ printf()□□□□□□□□□□□□□ datapoint□□□ ostream□□□□ printf()□□□□□□□□□□□□□□□

```
//: DATALOG.CPP -- Datapoint member functions
#include "..\5\datalog.h"
#include <iomanip.h>
#include <string.h>

tm datapoint::Time() { return Tm; }

void datapoint::Time(tm T) { Tm = T; }

const char* datapoint::latitude() {
  return Latitude;
}

void datapoint::latitude(const char* l) {
  Latitude[BSZ - 1] = 0;
  strncpy(Latitude, l, BSZ - 1);
}

const char* datapoint::longitude() {
  return Longitude;
}
```

```
void datapoint::longitude(const char* l) {
  Longitude[BSZ - 1] = 0;
  strncpy(Longitude, l, BSZ - 1);
}

double datapoint::depth() { return Depth; }

void datapoint::depth(double d) { Depth = d; }

double datapoint::temperature() {
  return Temperature;
}

void datapoint::temperature(double t) {
  Temperature = t;
}

void datapoint::print(ostream& os) {
  os.setf(ios::fixed, ios::floatfield);
  os.precision(4);
  os.fill('0'); // Pad on left with '0'
  os << setw(2) << Time().tm_mon << '\\'
     << setw(2) << Time().tm_mday << '\\'
     << setw(2) << Time().tm_year << ' '
     << setw(2) << Time().tm_hour << ':'
     << setw(2) << Time().tm_min << ':'
     << setw(2) << Time().tm_sec;
  os.fill(' '); // Pad on left with ' '
  os << " Lat:" << setw(9) << latitude()
     << ", Long:" << setw(9) << longitude()
     << ", depth:" << setw(9) << depth()
     << ", temp:" << setw(9) << temperature()
     << endl;
}
```

□ printf()□ □□□□□ setf()□ □□□□□□□□□□□□□□□□□□□□ precision()□ □ 4□ □□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ setw()□ □□□□□ 2□□ □□□□□□□□□□□□□□□□□□□□ setw()□ □□□□□□□□□ □□□□□□□□□□□ 10□ □□□□□□□□□□ “ 0”□ □□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□ ‘ *’ □□□ □□□□ `□□□□ ``□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

1. □□□□□□

□□□□□□□□□□□□□□ write()□ □□□□□□□□□□□□□□□□ ,□□□

datapoint::print()□□□□□ ASCII □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
//: DATAGEN.CPP -- Test data generator
#include <fstream.h>
#include <stdlib.h>
#include <string.h>
#include "..\5\datalog.h"

main() {
  ofstream data("data.txt");
  ofstream bindata("data.bin", ios::binary);
  time_t timer = time(NULL);  // Get time
  // Seed random generator:
  srand((unsigned)timer);
  for(int i = 0; i < 100; i++) {
    datapoint d;
    // Convert date/time to a structure:
    d.Time(*localtime(&timer));
    timer += 55; // Reading each 55 seconds
    d.latitude("45*20'31\"");
    d.longitude("22*34'18\"");
    // Zero to 199 meters:
    double newdepth  = rand() % 200;
    double fraction = rand() % 100 + 1;
    newdepth += double(1) / fraction;
    d.depth(newdepth);
    double newtemp = 150 + rand()%200; //Kelvin
    fraction = rand() % 100 + 1;
    newtemp += (double)1 / fraction;
    d.temperature(newtemp);
    d.print(data);
    bindata.write((unsigned char*)&d,
                  sizeof(d));
  }
}
```

□□DATA.TXT□□□□ASCII □□□□□□□□□□□□□□DATA.BIN□□□ios::binary□□□□□□□□□□□□□□□□□□□□

□□C□□□time(),□□□□□□□□□□□□□□□□□□□□□□time_t□□□□□1970□1□1□00□00□00 GMT□□□□□□□□□□□□□□C□□□scrand()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□tm□□□

```
struct tm {
   int tm_sec; // 0-59 seconds
```

```
    int tm_min; // 0-59 minutes
    int tm_hour; // 0-23 hours
    int tm_mday; // day of month
    int tm_mon; // 1-12 months
    int tm_year; // calendar year
    int tm_wday; // Sunday == 0, etc.
    int tm_yday; // 0-365 day of year
    int tm_isdst; // daylight savings?
};
```

□□□□□□□□□□□□ tm□ □□□□□□□□□□□ C□ localtime()□ □□□□□□□□□□□□□□□□□□□□□□□□ tm□ □□□□□□□ tm□ localtime()□ □□□□□□□□□□□□□ localtime()□ □□□□ localtime()□ □□□□□□□□□ datapoint□ □ tm struct□ □□□ localtime()□ □□□□□□□□□□□□□□□□□

`d.Time=*localtime(&timer)□`

□□□□□ timer□ □□ 55□ □□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ C□ rand()□ □□□□□□□□□□□□□□□ RAND_MAX□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ %□ □□□□□□□□□□□□□□□□□□□□□□□□□ rand()□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□ DATA.BIN□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□ RAM□ □□□□□□□□□□□□□□□□□□□□ write()□ □□□□□□□□□□□□□□□□□□ ——□□□□□□□□ unsigned char*□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□ datapoint□ □□□□□□□□□□□□□□ datapoint□ □□

2. □□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ DATA2.TXT□ □□□□□□□□□ DATA.TXT□ □□

```
//: DATASCAN.CPP -- Verify and view logged data
#include <fstream.h>
#include <strstrea.h>
#include <iomanip.h>
#include <assert.h>
#include "..\5\datalog.h"

main() {
  ifstream bindata("data.bin", ios::binary);
  assert(bindata); // Make sure it exists
  // Create comparison file to verify data.txt:
```

```
  ofstream verify("data2.txt");
  datapoint d;
  while(bindata.read((unsigned char*)&d,
                     sizeof d))
    d.print(verify);
  bindata.clear(); // Reset state to "good"
  // Display user-selected records:
  int recnum = 0;
  // Left-align everything:
  cout.setf(ios::left, ios::adjustfield);
  // Fixed precision of 4 decimal places:
  cout.setf(ios::fixed, ios::floatfield);
  cout.precision(4);
  for(;;) {
    bindata.seekg(recnum* sizeof d, ios::beg);
    cout << "record " << recnum << endl;
    if(bindata.read((unsigned char*)&d,
                    sizeof d)) {
      cout << asctime(&(d.Time()));
      cout << setw(11) << "Latitude"
           << setw(11) << "Longitude"
           << setw(10) << "Depth"
           << setw(12) << "Temperature"
           << endl;
      // Put a line after the description:
      cout << setfill('-') << setw(43) << '-'
           << setfill(' ') << endl;
      cout << setw(11) << d.latitude()
           << setw(11) << d.longitude()
           << setw(10) << d.depth()
           << setw(12) << d.temperature()
           << endl;
    } else {
      cout << "invalid record number" << endl;
      bindata.clear(); // Reset state to "good"
    }
    cout << endl
         << "enter record number, x to quit:";
    char buf[10];
    cin.getline(buf, 10);
    if(buf[0] == 'x') break;
    istrstream input(buf, 10);
    input >> recnum;
  }
}
```

ifstream bindata□□□□ DATA.BIN□□□□□□□□□□□□□□□□□□□□□

ios::nocreate[illegible] assert()[illegible] read()[illegible] datapoint d[illegible] datapoint[illegible] read()[illegible] bindata[illegible] failbit[illegible] while[illegible] “ [illegible] ” [illegible] clear()[illegible] failbit[illegible]

[illegible]

[illegible] recnum[illegible] 4[illegible]

[illegible]

```
record 0
Tue Nov 16 18:15:49 1993
Latitude    Longitude   Depth      Temperature
------------------------------------------------
45*20'31"   22*34'18"   186.0172   269.0167
```

[illegible] setw()[illegible] ‘ [illegible] ’[illegible] ‘ [illegible] ’[illegible]

[illegible] read()[illegible] else[illegible] failbit [illegible] clear()[illegible] read()[illegible]

[illegible] flat-file[illegible] DBMS——[illegible] AppleⅡ [illegible] BASIC[illegible] DBMS[illegible] C++[illegible]

6.11 [illegible]

[illegible]

6.12 [illegible]

1) [illegible] in [illegible] ifstream[illegible] os[illegible] ostrstream[illegible] rdbuf()[illegible] ostrstream[illegible] str()[illegible] os[illegible] char*[illegible] C toupper() [illegible] os[illegible]

2) [illegible]

3) [illegible] 1[illegible]

4) [illegible] grep[illegible]

[illegible] 7[illegible]

[illegible] const[illegible]

[illegible] C++[illegible] C[illegible] const[illegible] volatile,[illegible] const[illegible] “ [illegible] ” [illegible]

const[illegible] #defines[illegible]

7.1 [illegible]

[illegible] C[illegible] C++[illegible] const[illegible]

C[illegible]

```
#define BUFSIZE 100
```

BUFSIZE[illegible] “ [illegible] ”[illegible]

[illegible] BUFSIZE[illegible] C++[illegible] const[illegible]

```
const bufsize=100
```

[illegible]

```
const int bufsize=100
```

[illegible] bufsize[illegible]

```
char buf[bufsize]
```

[illegible] char[illegible] int[illegible] float[illegible] double[illegible] const[illegible] const[illegible] #define[illegible]

7.1.1 [illegible] const

[illegible] #define[illegible] const[illegible] const[illegible]

□ □ const□ C++□ □ const□ □ □ □ □ □ □ □ □ □ □ □ □ □ const□ □ const□ const□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ extern□ □ □ □ □ □ □ □

extern const bufsize□

□ □ □ □ □ extern□ const□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □□ □ □ C++□ □ □ □ □ □ □ □ const□ const□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

□ □ □ □ □ □ □ □ □ const□ const□ const□ □ □ cpp□ “ □ □ ” □ □ □ □ □ □ □ const□ □

7.1.2 const□ □ □ □

const□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ #defines□ const□ □

```
//: SAFECONS.CPP -- Using const for safety
#include <iostream.h>

const int i = 100;  // Typical constant
const int j = i + 10; // Value from const expr
long address = (long)&j; // Forces storage
char buf[j + 10]; // Still a const expression

main() {
  cout << "type a character & CR:";
  const char c = cin.get(); // Can't change
  const char c2 = c + 'a';
  cout << c2;
  // ...
}
```

□ □ □ □ □ □ i□ □ □ □ □ □ □ □ □ □ □ j□ □ i□ □ □ □ □ □ □ □ □ □ □ □ i□ □ □ □ □ □ j□ j□ □ □ □ □ □ □ □ □ □ □ □ j□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ j□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ j□ □ □ □ □ □ □ □ j□ buf□ □ □ □ □ □ j□

□ □ □ □ main()□ □ □ □ □ □ □ □ c□ □ □ □ □ □ const□ C□ □ □ □ □ □□ c2□ c□ ——□ □ □ #define□ □ □ □ □ □

□□□

□□□□□□□□□□□□□□□□□□□□□□□ const□□□

7.1.3 □□

const□ □□□const□ □□ “ □□□□□□□□□ ”□□□

```
//: CONSTAG.CPP -- Constants and aggregates

const int i[] = { 1, 2, 3, 4 };

//! float f[i[3]]; // Illegal

struct s { int i, j; };

const s S[] = { { 1, 2 }, { 3, 4 } };

//! double d[S[1].j]; // Illegal

main() {}
```

□□□ “ □□ ” □□□□□□□□□□□□□□□□□□□□□□

7.1.4 □ C□ □□□□

□□□□□□□□□ C++□ □□□□□□□ C□ □□□□□□□□□□□□□□□□□□□□□□□□□□□ C□ □□□□ C□ □ const□ □□ “ □□□□□□□□□□□□ ”□ □ C□ □□□□□□□□□□□□□□□□□□□□ C□ □□□□□ const□ □□□□□□□□□□□□ C□ □□□□□

```
const bufsize=100□
char buf[bufsize]□
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ bufsize□ □□□□□□□□□□□□ C□ □□□□□□□□□□□□□□ C□ □□□□□□□□□□□

```
const bufsize□
```

□□□□ C++□ □□□□□□ C□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ C□ □ const□ □□□□□□ C++□ □ const□ □□□□□□□□□□□□ C++□ □□□□□ C□ □□□□□□□□□ extern□ □□□□□□□□□

```
extern const bufsize;//declaration only
```

□□□□□□□□ C□□□□

□ C□ □□□□□□□ const□ □□ const□ □□□□□□□ C□ □□□□□□□□□□□□ #define□

7.2 □□

□□□□□□□□□□ const□□□□□ const□□

□□□□□□□□ const□□□□□□□□□ const□□□□□□□□□□□ const□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

7.2.1 □□ const□□□

□□ const□□□□“□□□”□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
const int* x□
```

□□□□□□□□□□□□□“ x□□□□□□□□□□□□ const int□”□□□□□□□□□□□□□ x□□□□□□□□□□□□□□□□□□□□□□ const□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ x□□□□□□□□□□□□□□□ const□□□□□□□□□□

```
int const* x□
```

□□□□□□□□□□□□□□□□□□□“ x□□□□□ int□ const□□”□□□□□□□□□□“ x□□□□□□□□ const□ int□□□□”□□ const□□□□□□ int□□□

7.2.2 const□□

□□□□□□□□□ const□□□□□□ const□□□□□□□ *□□□□□□

```
int d=1□
int* const x=&d□
```

□□□□□“ x□□□□□□□□□□□□ int□ const□□”□□□□□□□□□ const□□□ *x=2□

□□□□□□□□□□□□□□□□□□□□□□□ const□□□□□ const□□□

```
int d=1□
const int* const x=&d□  // (1)
int const* const x2=&d□  // (2)
```

□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□ const□□

- □□

□□□ ‘ *’“ □□ ”□□□□□□

```
int* u=&w□
```

int*□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□‘ *’□□ □□□ □□□

int* u=&w□ v=0□

□□□□□ int* u□□□□□□ int v□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□

7.2.3 □□□□□□□

C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ const□□ □□□□□□□ const□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ const□□□□□□□□□□ const□□□□□□□□□□□□□□□□□□□□□ const□□□ □□□ □□□□□□□ const□□□□□ const□□□□□□□□□□

```
int d = 1;
const int e = 2;
int* u = &d; // OK -- d not const
int* v = &e; // illegal -- e const
int* w = (int*)&e; // legal but bad practice
```

□□ C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□

• □□□□

□□□ const□□□□□□ const□□□□□□□□□□□□□□□□□□□□

char* cp="howdy"□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ “ howdy”□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□ □□□□

7.3 □□□□□□□□

□ const□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□ const□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□ const□□□□□□□□□□□□□□

7.3.1 □□ const□

□□□□□□□□□□□□□□□ const□□□□□□□□

```
void f1(const int i) {
  i++; // illegal -- compile-time error
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□x()□□

□□□□□const□□

□□□□□□□□□□□□□□□□const□□□□□□□□□□□const□□□□□□□□□□□□□□□□□□□□□□□□“□□”□□□□10□□&□□□□□□□□□□□□□□□□□□□□□

```
void f2(int ic) {
  const int& i = ic;
  i++;  // illegal -- compile-time error
}
```

□□□□□□□□□□□□□□□□□□□□□□const□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

7.3.2 □□const□

□□

const int g()□

□□□

□□□□□const□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
//: CONSTVAL.CPP -- Returning consts by value
// Has no meaning for built-in types

int f3() { return 1; }
const int f4() { return 1; }

main() {
  const int j = f3(); // Works fine
  int k = f4(); // But this works fine too!
}
```

□□□const□□□□□□□□□□□□

□□

```
//: CONSTRET.CPP -- Constant return by value
// Result cannot be used as an lvalue
class X {
  int i;
public:
  X(int I = 0) : i(I) {}
  void modify() { i++; }
};

X f5() {
  return X();
}

const X f6() {
  return X();
}

void f7(X& x) { // Pass by non-const reference
  x.modify();
}

main() {
  f5() = X(1); // OK -- non-const return value
  f5().modify(); // OK
  f7(f5()); // OK
  // Causes compile-time errors:
//!  f6() = X(1);
//!  f6().modify();
//!  f7(f6());
}
```

f5()□□□□□□const X□□□□□□f6()□□□□□const X□□□□□□const□□□const□

□□□□□□□□□□□□□□const□□□

□□f7()□□□□□□□□□□const□□□C++□□□□□□□□□□□□□□□10□□□□□□□□□□□□□□□□□□□□const□□□□□□□□□□□

• □□□□

□□——□□□

□□□□□□□□□□□□□□□□□□□□□□□□□

7.3.3 [illegible]

[illegible] const[illegible] const[illegible] const[illegible]

[illegible] const[illegible] const[illegible]

```
//: CONSTP.CPP -- Constant pointer arg/return

void t(int*) {}

void u(const int* cip) {
//!  *cip = 2; // Illegal -- modifies value
  int i = *cip; // OK -- copies value
//!  int* ip2 = cip; // Illegal: non-const
}

const char* v() {
  // Returns address of static string:
  return "result of function v()";
}

const int* const w() {
  static int i;
  return &i;
}

main() {
  int x = 0;
  int* ip = &x;
  const int* cip = &x;
  t(ip);  // OK
//!  t(cip); // Not OK
  u(ip);  // OK
  u(cip); // Also OK
//!  char* cp = v(); // Not OK
  const char* ccp = v(); // OK
//!  int* ip2 = w(); // Not OK
  const int* const ccip = w(); // OK
  const int* cip2 = w(); // OK
//!  *w() = 1; // Not OK
}
```

[illegible] t()[illegible] const[illegible] u()[illegible] const[illegible]

□□u()□□□□□□□□□□□□ const□□□□□□□□□□□□□□□□□□□□□□□□□ const□□□□□□□□□□□□□□□□ const□□□□□□□□□□□□□ const□□□

□□ v()□ w()□□□□□□□□□□□□□□ v()□□□□□□□□□□□□□□□□□□ const char*□□ v()□□□□□□□□□□

w()□□□□□□□□□□□□□□□□□□□□□□□□□□□□ v()□□□□□□□□□□□□□□□□□□□□□□ w()□□□

□□□ main()□□□□□□□□□□□□□□ t()□□□□□□ const□□□□□□□□□□□□□□□□□□□□□ const□□□□□□□□□□□□ t()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ u()□□□ const□□□□□□□□□□□□□□□□□□□□ const□□□□□□□ const□□□□□□□□□□

□□□□□□□□□ v()□□□□□□□□□□□ const□□□□□□□□□□□ w()□□□□□□□□□ const□□□□□□□□ const int* const□□□□□□□□□□□□□□ const int*□□□ const int*const□□□□□ const□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

- □□□□□□

□ C□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C++□□□□□□□□□□□□ C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□ const□□□ ——□□□ const□□

□□□ ——□□□□□□□□□□□□□□□□□□□□□□□□□□□□ C□□ const□□□□□□□□□□□□□□□□

```
//: CONSTTMP.CPP -- Temporaries are const

class X {};

X f() { return X(); } // Return by value

void g1(X&) {} // Pass by non-const reference
void g2(const X&) {} // Pass by const reference
main() {
  // Error: const temporary created by f():
//!  g1(f());
  // OK: g2 takes a const reference:
  g2(f());
}
```

□ □ f()□ □ □ X□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ f()□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ g1()□ g2()□ □ □ □ □ □□ g1()□ □ □ □ □ □ □ □ □ □ □ g1()□ □ □ □ □ □ □ const□ □ □ □ □ □ □ □ g2()□ □ □ □ □ □ □ □ □

7.4 □

□ □ □ □ □ □ □ const□ const□ ——□ □ □ □ □ □ □ □ □ □ □ □

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ const□□ □□ const□ const□ □ □ □ □

7.4.1 □ □ □ const□ enum

□ const□ □ #define□ size□ □ □ □ □ □ □ □ □ □ □ size□ #define□ #define□ □ □ □ □ □ □ □ □ □ □

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ const□ const□ □ □ □ C□ const□ □ □ □ “ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ”□ □

□ □ □ □ □ □ □ □ □ □ □ □ const□ const□ const□ const□ const□ □ □

1. □ □ □ □ □ □ □ □ □ □ □

□ □□ □ □ □ □ □ □ □ □ □ □ □ □ ——□ ——□ “ □ □ □ □ □ ”□ const□ □ □ □ □ □ □ □ □ □ □ □ □ const□ □ □ □ □ □

```
class fred {
  const size;
public:
  fred();
};

fred::fred() : size(100) {}
```

China-pub.com [illegible]

[illegible]

2. [illegible]“[illegible]”

[illegible]

```
class B {
  int i;
public:
  B(int I);
};

B::B(int I) : i(I) {}
```

[illegible] const[illegible]

[illegible]“[illegible]”[illegible]

float pi (3.14159)[illegible]

[illegible] integer[illegible]

```
class integer {
  int i;
public:
  integer(int I = 0);
};

integer::integer(int I) : i(I) {}
```

[illegible] integer[illegible]

integer I[100][illegible]

[illegible] for[illegible] memset()[illegible]

7.4.2 [illegible]

[illegible] const[illegible] const[illegible] C[illegible]

```
class bob {
  const size = 100;  // illegal
  int array[size];  // illegal
//...
```

[illegible] const[illegible]“[illegible] const”[illegible] enum[illegible]

□ □

```
class bunch {
  enum { size = 1000 };
  int i[size];
};
```

□ □ enum□ □

```
enum { one=1,two=2,three}□
```

□ □ □ □ enum□ three□ □ □ 3□

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ enum□ □ □ □

```
//: SSTACK.CPP -- Enums inside classes
#include <string.h>
#include <iostream.h>

class StringStack {
  enum { size = 100 };
  const char* stack[size];
  int index;
public:
   StringStack();
   void push(const char* s);
   const char* pop();
};

StringStack::StringStack() : index(0) {
  memset(stack, 0, size * sizeof(char*));
}

void StringStack::push(const char* s) {
  if(index < size)
    stack[index++] = s;
}
const char* StringStack::pop() {
  if(index > 0) {
    const char* rv = stack[--index];
    stack[index] = 0;
    return rv;
  }
  return 0;
}

const char* iceCream[] = {
  "pralines & cream",
  "fudge ripple",
  "jamocha almond fudge",
    "wild mountain blackberry",
    "raspberry sorbet",
    "lemon swirl",
    "rocky road",
    "deep chocolate fudge"
  };

  const ICsz = sizeof iceCream/sizeof *iceCream;

  main() {
    StringStack SS;
    for(int i = 0; i < ICsz; i++)
      SS.push(iceCream[i]);
    const char* cp;
    while((cp = SS.pop()) != 0)
      cout << cp << endl;
  }
```

□□push()□□□const char*□□□pop()□□□□const char*□stack□□const char*□□□□□□□□□□□□stringstack□□iceCream□□□□□□□□□□□□□□□□□□□□□□□□□Stringstack□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□enum□□□□C++□□□□□□□□static const□□□□□□□□□□□□□□□□□□□□□□□□□□□□□9□□□□

• □□□□□□□

C□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C++□□□

□□□C++□□□□□□□□C□□□□□□□□□□□□□□□□□□□a□□□□□color□□□□□□□□□□C□□□□a++□□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□color□□□int□□□□□1□□□int□□□□color□□C++□□□□□□□□□□□□□color□□□□int□□□□□□□□□□□blue□1□□□□□□□□□□□□□color□1□□□□□□□□□□□□□□□□□□□□enum□□□□□□□□□□□□□enum□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□

7.4.3 const□□□□□□

□□□const□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□const□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□const□□□□□□

```
const int i=1□
const blob B(2)□
```

□□□B□□blob□□□const□□□□□□□□□□□□□□□□□□“2”□□□□□□□□□□const□□□const□□□□□“□□”□□□

□□□□□□□□□□□const□□□□□□□□□□□□□□□□const□□□□□□□□□□□□□□□□□□□□const□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□const□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□const□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□const□□□□const□□□□□□□□□□□□□□□□□□□□□□□□□□const□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□const□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□const□□□□□□□□□□□□□const□□□□□□□□□□□□□□□□□

const□□□□□□□□□□□□□□□□□□□□□□□□□□const□□□□□□□□□□□□□

```
class X {
  int i;
public:
  int f() const;
};
```

□□□const□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

int X::f() const {return i□ }

□□f()□□□□□□□□□i□□□□□□□const□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□const□□□□□□□□□const□□□□□

□□□□□□□const□□const□□□□□□□□

```
//: QUOTER.CPP -- Random quote selection
#include <iostream.h>
#include <stdlib.h> // Random number generator
#include <time.h> // To seed random generator

class quoter {
  int lastquote;
public:
  quoter();
  int Lastquote() const;
  const char* quote();
};

quoter::quoter(){
  lastquote = -1;
  time_t t;
  srand((unsigned) time(&t)); // Seed generator
}
```

```
int quoter::Lastquote() const {
  return lastquote;
}

const char* quoter::quote() {
  static const char* quotes[] = {
    "Are we having fun yet?",
    "Doctors always know best",
    "Is it ... Atomic?",
    "Fear is obscene",
    "There is no scientific evidence "
    "to support the idea "
    "that life is serious",
  };
  const qsize = sizeof quotes/sizeof *quotes;
  int qnum = rand() % qsize;
  while(lastquote >= 0 && qnum == lastquote)
    qnum = rand() % qsize;
  return quotes[lastquote = qnum];
}

main() {
  quoter q;
  const quoter cq;
  cq.Lastquote(); // OK
//!  cq.quote(); // Not OK; non const function
  for(int i = 0; i < 20; i++)
    cout << q.quote() << endl;
}
```

[illegible] const[illegible] quote()[illegible] const[illegible] lastquote[illegible] Lastquote()[illegible] const[illegible] const[illegible] cq[illegible]

- [illegible] const

[illegible] const[illegible] const[illegible] const[illegible] const[illegible] const[illegible] const[illegible] const[illegible]

[illegible] “ [illegible] const”[illegible] this[illegible] this[illegible] const[illegible]

```
//: CASTAWAY.CPP -- "Casting away" constness

class Y {
  int i, j;
public:
  Y() { i = j = 0; }
  void f() const;
};

void Y::f() const {
//!    i++; // Error -- const member function
    ((Y*)this)->j++; //OK: cast away const-ness
}

main() {
  const Y yy;
  yy.f(); // Actually changes it!
}
```

this const mutable const

```
//: MUTABLE.CPP -- The "mutable" keyword

class Y {
  int i;
  mutable int j;
public:
  Y() { i = j = 0; }
  void f() const;
};

void Y::f() const {
//! i++; // Error -- const member function
    j++; // OK: mutable
}

main() {
  const Y yy;
  yy.f(); // Actually changes it!
}
```

const

7.4.4

const ROM

[illegible] const[illegible] ——[illegible] const[illegible] const[illegible] mutable[illegible] const[illegible] const[illegible]

1) class[illegible] struct[illegible]

2) [illegible] ,[illegible]

[illegible] const[illegible] ROM[illegible] ROM[illegible]

7.5 [illegible] volatile[illegible]

volatile[illegible] const[illegible] volatile[illegible] “ [illegible] ”[illegible] volatile[illegible] ——[illegible] “ [illegible] ”[illegible] volatile[illegible]

[illegible] const[illegible] volatile[illegible] const volatile[illegible]

```
//: VOLATILE.CPP -- The volatile keyword
class comm {
  const volatile unsigned char byte;
  volatile unsigned char flag;
  enum { bufsize = 100 };
  unsigned char buf[bufsize];
  int index;
public:
  comm();
  void isr() volatile;
  char read(int Index) const;
};

comm::comm() : index(0), byte(0), flag(0) {}

// Only a demo; won't actually work
// As an interrupt service routine:
void comm::isr() volatile {
  if(flag) flag = 0;
  buf[index++] = byte;
  // Wrap to beginning of buffer:
  if(index >= bufsize) index = 0;
}
```

```
char comm::read(int Index) const {
  if(Index < 0 || Index >= bufsize)
    return 0;
  return buf[Index];
}

main() {
  volatile comm Port;
  Port.isr(); // OK
//!  Port.read(0); // Not OK;
                // read() not volatile
}
```

□□const□□□□□□□□□□□□□□□□□□□□□□□ volatile□□□□□□□ □□volatile□□□□volatile□□□□□

□□isr()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ this□□□□□□□□□□□□□□□□□□□□ISR□□□□□□□□□□□□□□□□□□□□ isr()□□□□□□□□□□□□□□□□□□□□□

volatile□□□□const□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□c-v□□□□

7.6 □□

□□□const□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□ □□□□□□□□const correctness□□□□□□□□□□□const□□□□□□□□□□

□□□□const□□□□□□□C□□□□□□□□10□11□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□const□□□□□□

7.7 □□

1. □□□□□□□□□□fly()□□□bird□□□□□□□fly()□□□rock□□□□□□□ rock□□□□□□□□□□□□□□void*□□□□□□void*□□□□□□□bird*□□□□ □□□□□□□fly()□C□□□□□□□□□void*□□□C□□□□□□“□□”□□□□□□ □□□□□

2. □□□□□□const□□□□□□□□□□□□□□□□□□□□□□□const□□□□ □□□□□□□□□□□□□□□□□□□□□□

3. □□□□□□□□□□const□□const□□□□□□□□□□□const□□const□□□□ □□□□□□□□□□□□□□□□□□□□□

4. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

5. □□□□□□C□C++□□□□□□const□□□□□□□□□□□□□□□const□□□□□ □□□□□□□□□□□□C□C++□□□□□

□8□ □□□□

C++□□C□□□□□□□□□□□□□C++□□□□□□□C□□□□□□□□□□□□□

□C□□□□□□□□□□□□□□(macro)□□CALL□□□□□□□□□□□□RETURN□□

C++□□□□□□□□□□□□□□□□□□□□□C□□C++□□□□□□□□□□□□□□□(private)□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□C++□□□□□□(inline function)□□□□□□□□C++□□□□□□□□□□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□

8.1 □□□□□□□

□□□

□□□□□□□□□□□

```
#define f (x) (x+1)
```

□□□□□□□□□□□f□□□

```
f (1)
```

□□□□□□□□□□□□□□□□□□□

```
(x)  (x+1) (1)
```

□□□□□□□□□□□□□□f□□

```
f   (1)
```

□□□□□□□□□□□

```
(1 + 1)
```

□□

□□

```
#define floor(x,b) x>=b?0:1
```

□□□□□□□□□□□□

```
if(floor(a&0x0f,0x07)) // ...
```

□□□□□□

```
if(a&0x0f>=0x07?0:1)
```

□□&□□□□□>=□□

```
#define floor(x,b) ((x)>=(b)?0:1)
```

□□>=□□□□□□□+□ /□ --□□

□□

□□□□□□□□□□□□□□□□□□□□□□

```
#define band(x) (((x)>5 && (x)<10) ? (x) : 0)
```

□□□□□□“□□”□□

```
//: MACRO.CPP -- Side effects with macros
#include <fstream.h>
#define band(x) (((x)>5 && (x)<10) ? (x) : 0)

main() {
  ofstream out("macro.out");
  for(int i = 4; i < 11; i++) {
    int a = i;
    out << "a = " << a << endl << '\t';
    out << "band(++a)=" << band(++a) << endl;
    out << "\t a = " << a << endl;
  }
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
a = 4
      band(++a)=0
       a = 5
a = 5
      band(++a)=8
       a = 8
a = 6
      band(++a)=9
       a = 9
a = 7
      band(++a)=10
```

```
          a = 10
 a = 8
        band(++a)=0
          a = 10
 a = 9
        band(++a)=0
          a = 11
 a = 10
        band(++a)=0
          a = 12
```

[illegible]a[illegible]4[illegible]a[illegible]5[illegible]

[illegible]STDIO.H [illegible]putc( )[illegible]C[illegible]toupper( )[illegible]toupper(*p++)[1][illegible]

[illegible]

[illegible]C[illegible]C++[illegible]

```
class X {
  int i;
public:
#define val (X::i)  //Error
```

[illegible]public[illegible]

8.2 [illegible]

[illegible]C++[illegible]C++[illegible]

---

[1] [illegible]Andraw Koenig[illegible]C[illegible]Addision-Wesley,1989[illegible]

[illegible] inline[illegible]

```
inline int PlusOne(int x);
```

[illegible]

```
inline int PlusOne(int x) { return  ++x ;}
```

[illegible]

[illegible] + [illegible] CALL[illegible]

[illegible]——[illegible] static, [illegible]

8.2.1 [illegible]

[illegible] inline[illegible]

```
//: INLINE.CPP -- Inlines inside classes
#include <iostream.h>

class point {
  int i, j, k;
public:
  point() { i = j = k = 0; }
  point(int I, int J, int K) {
    i = I;
    j = J;
    k = K;
  }
  void print(const char* msg = "") const {
    if(*msg) cout << msg << endl;
    cout << "i = " << i << ", "
         << "j = " << j << ", "
         << "k = " << k << endl;
  }
};

main() {
  point p, q(1,2,3);
  p.print("value of p");
  q.print("value of q");
}
```

[illegible]

8.2.2 [illegible]

[illegible]——[illegible]

```
//: ACCESS.CPP -- Inline access functions

class access {
  int i;
public:
  int read() const { return i; }
  void set(int I) { i = I; }
};

main() {
  access A;
  A.set(100);
  int x = A.read();
}
```

[illegible] (private)[illegible] (private)[illegible] read( )[illegible] read( )[illegible] this[illegible] CALL[illegible]

[illegible] i[illegible] (public)[illegible], [illegible] i [illegible] i[illegible] i[illegible] int[illegible] float[illegible] int [illegible] int i[illegible]

- [illegible] (accessors)[illegible] (mutators)

[illegible]

□□□

```
//: RECTANGL.CPP -- Accessors & mutators

class rectangle {
  int Width, Height;
public:
  rectangle(int W = 0, int H = 0)
    : Width(W), Height(H) {}
  int width() const { return Width; } // Read
  void width(int W) { Width = W; } // Set
  int height() const { return Height; } // Read
  void height(int H) { Height = H; } // Set
};

main() {
  rectangle R(19, 47);
  // Change width & height:
  R.height(2 * R.width());
  R.width(2 * R.height());
}
```

□□□□□□□□□□□□□□□□□□□7□□□□□□□□13□□□□□□□□□□□Width□Height□□□□□□□□□□□□□□□□□□□□□□

□□C□□□□□□□□□□□□□□□Time□□

```
//: CPPTIME.H -- A simple time class
#ifndef CPPTIME_H_
#define CPPTIME_H_
#include <time.h>
#include <string.h>

class Time {
  time_t T;
  tm local;
  char Ascii[26];
unsigned char lflag, aflag;
void updateLocal() {
  if(!lflag) {
    local = *localtime(&T);
    lflag++;
  }
}
void updateAscii() {
  if(!aflag) {
      updateLocal();
      strcpy(Ascii, asctime(&local));
      aflag++;
    }
  }
public:
  Time() { mark(); }
  void mark() {
    lflag = aflag = 0;
    time(&T);
  }
  const char* ascii() {
    updateAscii();
    return Ascii;
  }
  // Difference in seconds:
  int delta(Time* dt) const {
    return difftime(T, dt->T);
  }
  int DaylightSavings() {
    updateLocal();
    return local.tm_isdst;
  }
  int DayOfYear() { // Since January 1
    updateLocal();
    return local.tm_yday;
  }
  int DayOfWeek() { // Since Sunday
  updateLocal();
  return local.tm_wday;
}
int Since1900() { // Years since 1900
  updateLocal();
  return local.tm_year;
}
int Month() { // Since January
  updateLocal();
  return local.tm_mon;
}
int DayOfMonth() {
  updateLocal();
  return local.tm_mday;
}
int Hour() { // Since midight, 24-hour clock
  updateLocal();
  return local.tm_hour;
```

```
    }
    int Minute() {
      updateLocal();
      return local.tm_min;
    }
    int Second() {
      updateLocal();
      return local.tm_sec;
    }
  };
  #endif // CPPTIME_H_
```

[illegible] C[illegible] Time[illegible] time_t T[illegible] tm local [illegible] ASCII[illegible] Ascii[illegible] time_t[illegible] updateLocal( )[illegible] updateAscii( )[illegible]

[illegible] mark( )[illegible] ASCII[illegible] ascii( )[illegible] updateAscii( )[illegible] ascii( )[illegible] updateAscaii( )[illegible] C[illegible]

[illegible] DaylightSaving( )[illegible] updateLocal( )[illegible] updateLocal( )[illegible]

[illegible]

```
//: CPPTIME.CPP -- Testing a simple time class
#include "..\7\cpptime.h"
#include <iostream.h>

main() {
  Time start;
  for(int i = 1; i < 1000; i++) {
    cout << i << ' ';
    if(i%10 == 0) cout << endl;
  }
  Time end;
  cout << endl;
  cout << "start = " << start.ascii();
  cout << "end = " << end.ascii();
  cout << "delta = " << end.delta(&start);
}
```

[illegible] Time[illegible] 2[illegible] Time[illegible]

8.3 [illegible]

[illegible]

[illegible]

[illegible] (this)[illegible]

8.3.1 [illegible]

[illegible]

[illegible]

[illegible]

[illegible] —— [illegible]

8.3.2 [illegible]

[illegible]

```
//: EVORDER.CPP -- Inline evaluation order

class forward {
  int i;
public:
  forward() : i(0) {}
  // Call to undeclared function:
  int f() const { return g() + 1; }
  int g() const { return i; }
};

main() {
  forward F;
  F.f();
}
```

[illegible] g( )[illegible] f( )[illegible] g( )[illegible]

[illegible] g( )[illegible] f( )[illegible] f( )[illegible] g( )[illegible] “ [illegible] ”[illegible]

8.3.3 [illegible]

[illegible]

```
//: HIDDEN.CPP -- Hidden activites in inlines
#include <iostream.h>

class member {
  int i, j, k;
public:
  member(int x = 0) { i = j = k = x; }
  ~member() { cout << "~member" << endl; }
};

class withMembers {
  member Q, R, S; // Have constructors
  int i;
public:
  withMembers(int I) : i(I) {} // Trivial?
  ~withMembers() {
    cout << "~withMembers" << endl;
  }
};

main() {
  withMembers WM(1);
}
```

□□ withMembers□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ Q□ P□ S□□□“□□”□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

8.4 □□□□

□□□ Dan Saks□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ *in situ*□□□□□□□□□□□□□(□□□□□□□)□□ inline□□□□□□□□□□□□.2.2□□ RECTANGL.CPP□□□□□□□

```
//: NOINSITU.CPP -- Removing in situ functions

class rectangle {
  int Width, Height;
public:
  rectangle(int W = 0, int H = 0);
  int width() const; // Read
  void width(int W); // Set
  int height() const; // Read
  void height(int H); // Set
};

inline rectangle::rectangle(int W, int H)
  : Width(W), Height(H) {
}

inline int rectangle::width() const {
  return Width;
}

inline void rectangle::width(int W) {
  Width = W;
}

inline int rectangle::height() const {
  return Height;
}

inline void rectangle::height(int H) {
  Height = H;
}
```

```
main() {
  rectangle R(19, 47);
  // Transpose width & height:
  R.height(R.width());
  R.width(R.height());
}
```

[illegible] inline[illegible] *In situ*[illegible] *in situ*[illegible]

8.5 [illegible]

[illegible] C[illegible] C++[illegible] 3[illegible] #[illegible]

```
#define DEBUG(X)  cout<<#X " = " << X << endl
```

[illegible]

```
#define TRACE(S) cout << #S << endl; S
```

#S[illegible] 2[illegible] S[illegible] for[illegible]

```
for (int i = 0 ; i < 100 ; i++ )
  TRACE(f(i)) ;
```

[illegible] TRACE( )[illegible] for[illegible]

[illegible]

[illegible]

```
#define FIELD(A) char* A##_string; int A##_size
class record {
  FIELD(one);
  FIELD(two);
  FIELD(three);
  // ...
};
```

[illegible] FIELD( )[illegible]

□ □

8.6 □ □ □ □ □ □ □

□ assert( )□ □ □ □ “ □ □ □ □ ”□ 17□ □

□ _FILE_□ _LINE_□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ assert( )□ □ □ □ □ assert( )□ assert( )□ □ □ □ □ □ □ □ □ □ □ □ □ □ assert( )□ □ □ □ □ □ □ □ □ □ □ □ #define NDEBUG□ □ □□

□ □“ Allege” □ assert □ □ □ □ □

```
//: ALLEGE.H -- Error checking
#ifndef ALLEGE_H_
#define ALLEGE_H_
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

inline void
allege_error(int val, const char * msg){
  if(!val) {
    fprintf(stderr, "error: %s\n", msg);
#ifdef NDEBUG
    exit(1);
#endif
  }
}

#define allege(expr, msg) \
{  allege_error((expr) ? 1 : 0, msg); \
  assert(expr); }

#define allegemem(expr) \
  allege(expr, "out of memory")

#define allegefile(expr) \
  allege(expr, "could not open file")
#endif // ALLEGE_H_
```

□ □ allege_error( )□ false□ □ □ □

□□□□□□ fprintf( )□□ iostreams□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□ exit(1)□□□□□□□□□

allege( )□□□□□□ if-then-else□□□□□□□□ expr□□□□□□□□□□ allege_error( )□□ □□ assert( )□□□□□□□□□□□□□□ assert( )□□□——□□□□□□□□□□□□□□ assert( )□□□□□□□

allegefile( )□□ allegemen( )□□□□□ allege( )□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□ ALLEGE.H□□□□□

```
//: ERRTEST.CPP -- Testing the allege() macro
// #define NDEBUG // turn off asserts
#include "..\allege.h"
#include <fstream.h>

main() {
  int i = 1;
  allege(i, "value must be nonzero");
  void* m = malloc(100);
  allegemem(m);
  ifstream nofile("nofile.xxx");
  allegefile(nofile);
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

//#define NDEBUG // turn off asserts

□□□□□□□□□□□□□ allege( )□□□□ assert( )□□□□□□□□□□□□□□□□ □□□□□□□□ assert( )□

8.7 □□

□□□

□□“□□□□□□□□□□□□□□□□□□□□□□□□”

8.8 □□

1. □□7□□□2□□□□□□□□□□□□□□□□□□□□□ Print( )□□□□□□□□□□□□□□□□□□□□□□□□□□

2. □□□3□□ NESTFRND.CPP□□□□□□□□□□□□□□□□□□□□□□□□□□

in situ□□□□□□□□□ initialize( )□□□□□□□□□

3. □□□6□ NL.CPP□□□□□□□□□□□□ nl□□□□□□□□

4. □□□□□ A□□□□□□□□□□□□□□□□□□□□□□ B, □ A□□□□□□□ B□□□□□□□ B□□□□□□□□□□□□□□ B□□□□□□□□□□□□□□□

5. □□□4□□□□□□□□□□□ Time□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□

[illegible]

[illegible] 9[illegible]

[illegible] C++ [illegible]

static[illegible]“[illegible]”[illegible] C[illegible] C++[illegible] static[illegible]“[illegible]”[illegible]“[illegible]”[illegible]

[illegible] static[illegible] C++[illegible] C[illegible]

9.1 [illegible] C[illegible]

[illegible] C[illegible] C++[illegible] static[illegible]

1) [illegible]

2) [illegible] C++[illegible] static[illegible]

[illegible] static[illegible] C[illegible]

9.1.1 [illegible]

[illegible]

[illegible] C[illegible] C++[illegible] static[illegible]

```
//: STATFUN.CPP -- Static vars inside functions
#include <iostream.h>
#include "..\allege.h"

char onechar(const char* string = 0) {
  static const char* s;
  if(string) {
    s = string;
    return *s;
  }
  else
    allege(s, "un-initialized s");
  if(*s == '\0')
    return 0;
  return *s++;
}

char* a = "abcdefghijklmnopqrstuvwxyz";

main() {
  // Onechar(); // causes allege()
  onechar(a); // Initializes s to a
  char c;
  while((c = onechar()) != 0)
    cout << c << endl;
}
```

static char* s onechar() onechar() s onechar() 0 string s ——

 onechar() s s

static char* s=0

 onechar() s if(!s)

 s

1.

```
//: FUNOBJ.CPP -- Static objects in functions
#include <iostream.h>

class X {
  int i;
public:
  X(int I = 0) : i(I) {} // Default
  ~X() { cout << "X::~X()" << endl; }
};
void f() {
  static X x1(47);
  static X x2; // Default constructor required
}

main() {
  f();
}
```

□□□f()□□□□□□□□□X□□

2.□□□□□□□□□

□□□main()□□□□□□□□□□□C□□□exit()□□□□□□□□□□□□□□main()□□□□□□□□□exit()□□□□□□□□□□□□□□□□□□□□exit()□□□□□□□□□□□□□□□□□□□□□□□□□□C□□□abort()□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□C□□□atexit()□□□□□□□□main()□□□□exit()□□□□□□□□□□□□□□□□□□□main()□□□exit()□□□□atexit()□□□□□□□□□□□□□□□□□□□□□□□

□□main()□□□

```
//: STATDEST.CPP -- Static object destructors
#include <fstream.h>
ofstream out("statdest.out"); // Trace file

class obj {
  char c; // Identifier
public:
  obj(char C) : c(C) {
    out << "obj::obj() for " << c << endl;
  }
  ~obj() {
    out << "obj::~obj() for " << c << endl;
  }
};

obj A('A'); // Global (static storage)
// Constructor & destructor always called

void f() {
  static obj B('B');
}

void g() {
  static obj C('C');
}

main() {
  out << "inside main()" << endl;
  f(); // Calls static constructor for B
  // g() not called
  out << "leaving main()" << endl;
}
```

[illegible] obj[illegible] c[illegible] A[illegible] obj[illegible] main()[illegible] f()[illegible] obj[illegible] B[illegible] g()[illegible] C[illegible]

[illegible] main()[illegible] f()[illegible]

```
obj::obj() for A
inside main()
obj::obj() for B
leaving main()
obj::~obj() for B
obj::~obj() for A
```

[illegible] A[illegible] main()[illegible] B[illegible] f()[illegible] main()[illegible] g()[illegible] B[illegible] C[illegible] g()[illegible] f()[illegible]

[illegible] ofstream[illegible] out[illegible] extern[illegible] out[illegible]

[illegible] C++[illegible] main()[illegible] main()[illegible] main()[illegible] C[illegible]

9.1.2 [illegible]

[illegible]

[illegible]

[illegible] static[illegible] .CPP[illegible]

[illegible] inline function[illegible] C++[illegible] C[illegible] /[illegible]

• □ □ □ □

□ □ □ □ □ □ □ static□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □□ □

```
int a=0;
```

□ a□ main()□ □ □ □ □ a□ □ □ □ □ □ □ □ □ □ □ a□ static□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ extern□ □

```
extern int a=0;
```

□ □ □ □ □ □ □ :

```
static int a=0;
```

□ □ □ □ □ □ □ □ a□ □ □ □ □ □ □ a□ ——□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ static□ □ extern□

□ □ □ □ □ □ □ □ □ static□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ extern□ □ □ □ □ □ □□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

□ □ □ □ □ static□ extern□ □

```
extern void f()□
```

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

```
void f()□
```

□ □ □ □ □

```
static void f()□
```

□ □ □ □ f()□ □

9.1.3 □ □ □ □ □ □ □ □ □ □

□ □ □ □ □ static□ extern□ auto,□ auto□ □ □ □ □ □ □ □ □ □ □ register□ register□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ register□ □

9.2 □ □ □ □

□ static□ □□ typedef□ □ □ □ □ □ □ □ □ □ □ □□ □

□ □ □ □ □ C++□ □□ class,struct,enum □ union□ ,□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

□ namespace□□□□□□□□□□□□□□□□□

9.2.1 □□□□□□□□

□□□□□□□□□□□□□□□□□□□

```
namespace MyLib {
  // Declarations
}
```

□□□□□□□□□□□□□□□□□□□□□□□□ namespace□ class□ struct□ union □ enum□□□□□□□□

1) namespace□□□□□□□□□□□□□□□□□□□□□□

2) □ namespace□□□□□□□□□□□□□□□□□□□□□□

3) □□ namespace□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
//: HEADER1.H
namespace MyLib {
  extern int X;
  void f();
  // ...
}
//: HEADER2.H
// Add more names to MyLib
namespace MyLib { // NOT a redefinition!
  extern int Y;
  void g();
  // ...
}
```

4) □□ namespace□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□

```
namespace BobsSuperDuperLibrary {
  class widget { /* ... */ };
  class poppit { /* ... */ };
  // ...
}
// Too much to type! I'll alias it:
namespace Bob = BobsSuperDuperLibrary;
```

5) □□□□□□□□□□□□□□□□□□□□□

1. □□□□□□□□

□□□□□□□□□□□□□□□□□□□□——□ namespace□□□□□□□□□□□

```
namespace {
  class arm  { /* ... */ };
  class leg  { /* ... */ };
  class head { /* ... */ };
  class robot {
    arm Arm[4];
    leg Leg[16];
    head Head[3];
    // ...
  } Xanthan;
  int i, j, k;
}
```

□□□static□□□□□□□□□□□□□□

2.□□

□□□□□□□□□□□□□□□□□□□friend□□□

```
namespace me {
  class us {
    //...
    friend you();
  };
}
```

□□□□you()□□□□□□□me□□□□□□

9.2.2 □□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□using□□□□

1.□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
namespace X {
  class Y {
    static int i;
  public:
    void f();
  };
  class Z;
  void foo();
}

int X::Y::i = 9;

class X::Z {
  int u, v, w;
public:
  Z(int I);
  int g();
};
```

```
X::Z::Z(int I) { u = v = w = I; }
int X::Z::g() { return u = v = w = 0; }

void X::foo() {
  X::Z a(1);
  a.g();
}
```

□□□□□□□□□□□□□□□□□□□

2. using□□

□ using □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ using□ namespace□□□□□□□□□□ using □□□ using □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
namespace math {
  enum sign { positive, negative };
  class integer {
    int i;
    sign s;
  public:
    integer(int I = 0)
      : i(I),
        s(i >= 0 ? positive : negative)
    {}
    sign Sign() { return s; }
    void Sign(sign S) { s = S; }
    // ...
  };
  integer A, B, C;
  integer divide(integer, integer);
  // ...
}
```

□□□□□□□□□□□ math□□□□□□□□□□□□□□□□□□□□□

```
void arithmetic() {
  using namespace math;
  integer X;
  X.Sign(positive);
}
```

□□□□ using□□□□□□□□□□□□□□□□□□□□□□

using □□□□□□□□□□□□□□□□□□□□ using□□□□□□□□□□□□□□□□□□ using□□□□□□□□□□□□ using □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
void q() {
  using namespace math;
  integer A; // Hides math::A;
```

□□

```
  A.Sign(negative);
  math::A.Sign(positive);
}
```

□□□□□□□□□□

```
namespace calculation {
  class integer {};
  integer divide(integer, integer);
  // ...
}
```

□□□□□□□□ using□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□ using□□□□□□

```
void s() {
  using namespace math;
  using namespace calculation;
 // Everything's ok until:
 divide(1, 2); // Ambiguity
}
```

□□□□□□□□□□□□□□□ using□□□□□□□□□□□□□□□□□□□

3. using□□

□□□ using□□□□□□□□□□□□□□□□□□□□□ using□□□ using□□□□□

```
namespace U {
  void f();
  void g();
}

namespace V {
  void f();
  void g();
}

void func() {
  using namespace U; // Using directive
  using V::f; // Using declaration
  f(); // Calls V::f();
  U::f(); // Must fully qualify to call
}
```

using□□ using□□□□□□□□□□□□□□□□□□□

□□□ using□□□□□□□□□□□□□□□□□□ using□□□□□□□□□□□ □□ using□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□

using□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
namespace Q {
  using U::f;
  using V::g;
  // ...
}

void m() {
  using namespace Q;
  f(); // calls U::f();
  g(); // calls V::g();
}
```

□□using□□using□□□□□□□□□□□□□□□□□□

9.3 C++□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□C□□□

□□public□□□□□□private□□□□□□□protected□□□□□□

9.3.1 □□□□□□□□□□□

□□□

□□□

```
class A {
  static int i;
public:
  //...
};
```

□□□□□□□□□□

```
int A::i=1;
```

□□□□□□□□□□□□□□□□□□□

int i=1;

[illegible] i[illegible]

[illegible] A::i[illegible]

“ =” [illegible] —— [illegible]

[illegible]

```
//: STATINIT.CPP -- Scope of static initializer
#include <iostream.h>

int x = 100;

class withStatic {
  static int x;
  static int y;
public:
  void print() const {
    cout << "withStatic::x = " << x << endl;
    cout << "withStatic::y = " << y << endl;
  }
};

int withStatic::x = 1;
int withStatic::y = x + 1;
// WithStatic::x NOT ::x

main() {
  withStatic WS;
  WS.print();
}
```

[illegible] withStatic::[illegible] withStatic[illegible]

1. [illegible]

[illegible]

```
//: STATARRY.CPP -- Initializing static arrays

class Values {
  static const int size;
  static const float table[4];
  static char letters[10];
};
```

```
const int Values::size = 100;

const float Values::table[4] = {
  1.1, 2.2, 3.3, 4.4
};

char Values::letters[10] = {
  'a', 'b', 'c', 'd', 'e',
  'f', 'g', 'h', 'i', 'j'
};

main() {}
```

[illegible]

2. [illegible]

[illegible]7[illegible] “enum hack”[illegible]

[illegible]

```
class X {
  static const int size;
  int array[size];
public:
  // ...
};

const int X::size = 100; // definition
```

[illegible]

[illegible] enum[illegible]

9.3.2 [illegible]

[illegible]——[illegible]

□□□□□□□□□□□

```
//: LOCAL.CPP -- Static members & local classes
#include <iostream.h>

// Nested class CAN have static data members:
class outer {
  class inner {
    static int i; // OK
  };
};

int outer::inner::i = 47;

// Local class cannot have static data members:
void f() {
  class foo {
  public:
//! static int i;  // Error
    // (how would you define i?)
  } x;
}

main() {}
```

□□□

9.3.3 □□□□□□

□□□

□□□ this□ □□□□□□□□□□□□□□□□□□□□□□□□□□ this□ □□ this□ □□□□□□□□□□□□□□□□□□□□□□

□ static□ □□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
//: SFUNC.CPP -- Static member functions

class X {
  int i;
  static int j;
public:
  X(int I = 0) : i(I) {
     // Non-static member function can access
     // Static member function or data:
    j = i;
  }
  int val() const { return i; }
  static int incr() {
    //! i++; // Error: static member function
    // Cannot access non-static member data
    return ++j;
  }
  static int f() {
    //! val(); // Error: static member function
    // Cannot access non-static member function
    return incr(); // OK -- calls static
  }
};

int X::j = 0;

main() {
  X x;
  X* xp = &x;
  x.f();
  xp->f();
  X::f(); // Only works with static members
}
```

□□□□□□□□□□this□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□this□□□□

□□□main()□□□

□□egg□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□egg□□□

```
//: SELFMEM.CPP -- Static member of same type
// Ensures only one object of this type exists.
// Also referred to as a "singleton" pattern.
#include <iostream.h>

class egg {
  static egg E;
  int i;
  egg(int I) : i(I) {}
public:
  static egg* instance() { return &E; }
  int val() { return i; }
};

egg egg::E(47);

main() {
//!  egg x(1); // error -- can't create an egg
  // You can access the single instance:
  cout << egg::instance()->val() << endl;
}
```

E□□□

9.4 □□□□□□□□□□

□□

□□

```
// first file
#include <fstream.h>
ofstream out("out.txt");
```

□□□□□□□□□□□□□□□□□□□ out □□□

```
// second file
#include <fstream.h>
extern ofstream out;
class oof {
public:
  oof() { out << "barf"; }
} OOF;
```

□□□oof□□□□□□□ out□□□□□□□□ out□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ main()□□□□□□□□□□□□□□□□□□□□□□

□ ARM [1] □□□□□□□□□□□□□□□□□□□□□

```
extern int y;
int x = y + 1;
```

---

[1] The Annotated C++ Reference Manual,Bjarne Stroustrup□ Margaret Ellis□ □ 1990□ □ pp.20-21□

□□□□□□□

```
    extern int x;
    int y = x + 1;
```

□□fstream out□□y□□□□□□□□□□x□□1□□□y□□□□□□□□2□□□□□□□□□□□□□□□□□□□x□□□□□□□□,y□□□□□1□□□x□□□□□2□

□□

□□□

□□□□□□□□□□□□□□

1) □□□□□□□□□□□□□□□□□□□□□□□□□□□

2) □□

3) □□ Jerry Schwarz□□□iostream□□□□cin,cout□cerr□□□□□□□□□□□□□□□□□□□□□

□□

```
//: DEPEND.H -- Static initialization technique
#ifndef DEPEND_H_
#define DEPEND_H_
#include <iostream.h>
extern int x; // Delarations, not definitions
extern int y;

class initializer {
  static int init_count;
public:
  initializer() {
    cout << "initializer()" << endl;
    // Initialize first time only
    if(init_count++ == 0) {
      cout << "performing initialization"
           << endl;
      x = 100;
      y = 200;
    }
  }
~initializer() {
  cout << "~initializer()" << endl;
```

China-pub.com [illegible]

```
    // Clean up last time only
    if(--init_count == 0) {
      cout << "performing cleanup" << endl;
      // Any necessary cleanup here
    }
  }
};

// The following creates one object in each
// file where DEPEND.H is included, but that
// object is only visible within that file:
static initializer init;

#endif // DEPEND_H_
```

x[illegible] y[illegible] initializer init [illegible] static[illegible]

[illegible] x[illegible] y[illegible] init_count[illegible]

```
//: DEPDEFS.CPP -- Definitions for DEPEND.H
#include "depend.h"
// Static initialization will force
// all these values to zero:
int x;
int y;
int initializer::init_count;
```

[illegible] init[illegible]

```
//: DEPEND.CPP -- Static initialization
#include "depend.h"
```

[illegible]

```
//: DEPEND2.CPP -- Static initialization
#include "depend.h"

main() {
  cout << "inside main()" << endl;
  cout << "leaving main()" << endl;
}
```

[illegible] DEPEND.H[illegible] init_count[illegible] ~initializer()[illegible]

□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□initializer□□ initializer()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C++□□□□ new□ delete□ □ 12□ □□□□

9.5 □□□□□□

□□C++□□□□□□□□□□□□C□□□□□□□□□□□□□□□□□□□□□C□□□

```
float f(int a,char b);
```

C++□□□□□□□□□□□□□□□□□□ _f_int_int□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C□□□□□□□□□□□□□□□□□□□□□□□□□□ _f□□□□□□□□□□□□□□□□□C++□ f□□□□□□

C++□□□□□□□□□□□□□□□□□□□□□□□□□□□ extern□□□□□□□□□□ extern□□

```
extern "C" float f(int a,char b);
```

□□□□□□□□ f()□ C□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“ C”□ “ C++”□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□

```
extern "C" {
  float f(int a, char b);
  double d(int a, char b);
}
```

□□□□□□□□□

```
extern "C" {
#include "myheader.h"
}
```

□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ C□ C++□□□□□□□□□□□□□

□□□□□C++□□□“ C”□“ C++”□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

9.6 □□

static□□□

□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□ static□□□

9.7 □□

1. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

const int □□□□□□□□□□□□□□□□□□□□□□□□□ static int □□□□□□□□□□□□□□□□□□□□□ inline□ □□□□□□□□□□ inline□ □□ print()□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□

2. STATDEST.CPP□ □□ main()□ □□□□□□□□ f()□ g()□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

3. □ STATDEST.CPP□ □□ out□ □□□□□□□ extern□ □□□□□□□□□□ A□ □□□□□□ obj□ □□□□ out□ □□

4. □□□□□□□□□□□□□□□□□□□□□□□ exit()□ □□□□□□□□□□□□□□□□□□□□□□□

5. □□□7□ □ VOLATILE.CPP□ □ comm::isr()□ □□□□□□□□□□□□□

[illegible] 10[illegible] [illegible]

[illegible] C++[illegible]

[illegible] Pascal[illegible] C++[illegible] Algol[illegible] C++[illegible] 11[illegible]

[illegible] C[illegible] C++[illegible] C++[illegible] (copy-constructor)[illegible] (&)[illegible]

[illegible] C++[illegible] (pointer-to-member)[illegible]

10.1 C++[illegible]

C[illegible] C++[illegible] C++[illegible] void*[illegible] C[illegible] void*[illegible]

```
bird*  b;
rock*  r;
void*  v;
v = r;
b = v;
```

C++[illegible] 18[illegible] C++[illegible]

10.2 C++[illegible]

[illegible] (&)[illegible]

```
int  x;
int  & r = x;
```

[illegible]

```
int  & q = 12;
```

[illegible] 12[illegible]

```
int x=0;
int & a = x;
a++;
```

[illegible] a[illegible] x[illegible]

□□□□□□□□□□□□

□□□□□□□□□□□□

1) □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

2) □□

3) □□□□NULL□□□□□□□□□□□□□□□□□□□□□□□

10.2.1 □□□□□□

□□□

□□

□□□□□□□□

```
//: REFRNCE.CPP -- Simple C++ references

int* f(int* x) {
  (*x)++;
  return x; // Safe; x is outside this scope
}

int& g(int& x) {
  x++; // Same effect as in f()
  return x; // Safe; outside this scope
}

int& h() {
  int q;
//!  return q;  // Error
  static int x;
  return x; // Safe; x lives outside scope
}

main() {
  int A = 0;
  f(&A); // Ugly (but explicit)
  g(A);  // Clean (but hidden)
}
```

□□□f( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ g ( )□□□□□□□□□□□□□□□□□□□□□□□□

1. □□□□

□□□REFRNCE.CPP□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□g ( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□

□□□

```
//: PASCONST.CPP -- Passing references as const

void f(int&) {}
void g(const int&) {}

main() {
//!  f(1); // Error
  g(1);
}
```

□□f(1)□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□int□□□□□□□□□□□□□□□□ 1□□□

2. □□□□

□C□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
void f (int**);
```

□□□□□□□□□□□□□□□□□□□□□□

```
int I = 47;
int* ip = &I;
f (&ip);
```

□□C++□□□

```
//: REFPTR.CPP -- Reference to pointer
#include <iostream.h>

void increment(int*& i) { i++; }

main() {
  int* i = 0;
  cout << "i = " << i << endl;
  increment(i);
  cout << "i = " << i << endl;
}
```

10.2.2 □□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

[illegible]

[illegible]

[illegible]

10.3 [illegible]

[illegible]C++[illegible]X(X&)[illegible]“ X[illegible]X”[illegible]

10.3.1 [illegible]

[illegible] C[illegible]

```
int f (int x, char c);
int g = f (a,b);
```

[illegible] char[illegible] int[illegible] float[illegible] double[illegible]

[illegible] f ( )[illegible]

```
push b
push a
call f ( )
add sp,4
mov g, register a
```

[illegible]——b[illegible] a[illegible] _b[illegible] _a[illegible] g[illegible] f( )[illegible] name-mangling[illegible]“ [illegible] a” [illegible] CPU[illegible]

[illegible]C[illegible] C++[illegible] add sp,4[illegible]——[illegible]

f( )[illegible]

1. [illegible]

[illegible]

[illegible]

```
//: PASSTRUC.CPP -- Passing a big structure

struct big {
  char buf[100];
  int i;
  long d;
} B, B2;

big bigfun(big b) {
  b.i = 100; // Do something to the argument
  return b;
}

main() {
  B2 = bigfun(B);
}
```

[illegible] “helper” [illegible] main() [illegible] bigfun() [illegible] B [illegible] B [illegible] helper [illegible]

[illegible] PASSTRUC.CPP [illegible] B2 [illegible]

2. [illegible]

[illegible] “[illegible]” [illegible] CALL [illegible] CPU [illegible] RETURN [illegible] CALL [illegible]

[illegible]
[illegible]
[illegible]

[illegible] (function frame) [illegible]

3. [illegible]

[illegible] C [illegible] C++ [illegible] ISR [illegible] ISR [illegible]

[illegible] ISR [illegible] void [illegible] CPU [illegible] ISR [illegible]

[illegible] RETURN [illegible] RETURN [illegible] ISR [illegible]

[illegible] C [illegible] C++ [illegible] C++ [illegible]

[illegible]

[illegible] PASSTRUC.CCP [illegible] main( ) [illegible] bigfun( ) [illegible] B2 [illegible] bigfun( ) [illegible]

4. [illegible] bitcopy [illegible]

[illegible] C [illegible] C++ [illegible]

[illegible] 9 [illegible] (static) [illegible]

```
//: HOWMANY.CPP -- Class counts its objects
#include <fstream.h>
ofstream out("howmany.out");

class howmany {
  static int object_count;
public:
  howmany() {
    object_count++;
  }
  static void print(const char* msg = 0) {
    if(msg) out << msg << ": ";
    out << "object_count = "
         << object_count << endl;
  }
  ~howmany() {
    object_count--;
```

```
        print("~howmany()");
      }
    };

    int howmany::object_count = 0;

    // Pass and return BY VALUE:
    howmany f(howmany x) {
      x.print("x argument inside f()");
      return x;
    }

    main() {
      howmany h;
      howmany::print("after construction of h");
      howmany h2 = f(h);
      howmany::print("after call to f()");
    }
```

howmany[illegible] int[illegible] print( )[illegible]

[illegible]

```
after construction of h: object_count = 1
x argument inside f(): object_count = 1
~howmany(): object_count = 0
after call to f(): object_count = 0
~howmany(): object_count = -1
~howmany(): object_count = -2
```

[illegible] h[illegible] 1[illegible] f( )[illegible] 2[illegible] h2[illegible] 0[illegible]

[illegible] f( )[illegible] h[illegible] C[illegible] C++ howmany[illegible]

[illegible] f( )[illegible] object_count[illegible] object_count[illegible] 0[illegible] h2[illegible] h[illegible] h2[illegible] object_count[illegible]

10.3.2 [illegible]

[illegible]

□□□
□□□□□□□□□□□□□□□□□□□□□□□□□

```
howmany h2 = f(h);
```

□□□□□□□□□h2□□□□□f( )□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□

□□□□□□□□□□□□□□bitcopy□□□□□□□□□□□□□□□□□□□□□howmany□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□(bitcopy)□□□X(X&)□□□□□□X□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□(bitcopy)□□

□□□□□□□□HOWMANY.CPP□□□□□□

```
//: HOWMANY2.CPP -- The copy-constructor
#include <fstream.h>
#include <string.h>
ofstream out("howmany2.out");

class howmany2 {
  enum { bufsize = 30 };
  char id[bufsize]; // Object identifier
  static int object_count;
public:
  howmany2(const char* ID = 0) {
    if(ID) strncpy(id, ID, bufsize);
    else *id = 0;
    ++object_count;
    print("howmany2()");
  }
  // The copy-constructor:
  howmany2(const howmany2& h) {
    strncpy(id, h.id, bufsize);
    strncat(id, " copy", bufsize - strlen(id));
    ++object_count;
    print("howmany2(howmany2&)");
  }
  // Can't be static (printing id):
    void print(const char* msg = 0) const {
      if(msg) out << msg << endl;
      out << '\t' << id << ": "
          << "object_count = "
          << object_count << endl;
    }
    ~howmany2() {
      --object_count;
      print("~howmany2()");
    }
  };

  int howmany2::object_count = 0;

  // Pass and return BY VALUE:
  howmany2 f(howmany2 x) {
    x.print("x argument inside f()");
    out << "returning from f()" << endl;
    return x;
  }

  main() {
    howmany2 h("h");
    out << "entering f()" << endl;
    howmany2 h2 = f(h);
    h2.print("h2 after call to f()");
    out << "call f(), no return value" << endl;
    f(h);
    out << "after call to f()" << endl;
  }
```

□□□□□□□□□□□□□□□□□□□□□□□□□ id□□C□□□ strncpy( )□□□□□□□□ id□ strncpy( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□ howmany2(howmany2&)□□ id□ id□□□□□□“ copy”□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ C□□□ strncat( )□□□□□ id□□□□□□□□□□□□□□□□

□□□

print( )□□□□□□□□□□□□□□□□□□□□□□□□□□□ print( )□□□□□□□□□□□ id□□□□□□□□ static□□□□□

□ main( )□□□□□□□□□□□□□□□□□□□□ f( ) □□□□□□□□□□□□□ C□□□

□□

□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□ iostreams□ □□□□□□□

```
//: LINENUM.CPP -- Add line numbers
#include <fstream.h>
#include <strstrea.h>
#include <stdlib.h>
#include "..\allege.h"

main(int argc, char* argv[]) {
  if(argc < 2) {
    cerr << "usage: linenum file\n"
         << "adds line numbers to file"
         << endl;
    exit(1);
  }
  strstream text;
  {
    ifstream in(argv[1]);
    allegefile(in);
    text << in.rdbuf(); // Read in whole file
  } // Close file
  ofstream out(argv[1]); // Overwrite file
  const bsz = 100;
  char buf[bsz];
  int line = 0;
  while(text.getline(buf, bsz)) {
    out.setf(ios::right, ios::adjustfield);
    out.width(2);
    out << ++line << ") " << buf << endl;
  }
}
```

□□□□□□□ strstream□ □□□□□□□□□□□□ifstream□ □□□□□□□□□□□□ □□□□□□□□ ofstream, ofstream□ □□□□□□ getline( )□ strstream□ □□□□□□□□□ □□□□□□□

□□□□□□□□□ 2□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□ 2□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□

□ LINENUM.CPP□ □□□ HOWMANY2.OUT□ □□□□□□

```
1) howmany2()
2)   h: object_count = 1
3) entering f()
4) howmany2(howmany2&)
5)   h copy: object_count = 2
6) x argument inside f()
```

```
  7)    h copy: object_count = 2
  8) returning from f()
  9) howmany2(howmany2&)
 10)    h copy copy: object_count = 3
 11) ~howmany2()
 12)    h copy: object_count = 2
 13) h2 after call to f()
 14)    h copy copy: object_count = 2
 15) call`f(), no return value
 16) howmany2(howmany2&)
 17)    h copy: object_count = 3
 18) x argument inside f()
 19)    h copy: object_count = 3
 20) returning from f()
 21) howmany2(howmany2&)
 22)    h copy copy: object_count = 4
 23) ~howmany2()
 24)    h copy: object_count = 3
 25) ~howmany2()
 26)    h copy copy: object_count = 2
 27) after call to f()
 28) ~howmany2()
 29)    h copy copy: object_count = 1
 30) ~howmany2()
 31)    h: object_count = 0
```

[illegible] h[illegible] 1[illegible] f( )[illegible] f( )[illegible] h[illegible] “h[illegible]”[illegible] 2[illegible]

[illegible] 8[illegible] f( )[illegible] “h[illegible]” [illegible] h2[illegible] h2[illegible] f( )[illegible] 9[illegible] h2[illegible] “h[illegible]”[illegible] f( ) [illegible] 3[illegible] “h[illegible]” [illegible] 13[illegible] f( )[illegible] 2[illegible] h[illegible] h2 [illegible] h2[illegible] “h[illegible]”[illegible]

- [illegible]

15[illegible] f(h)[illegible] 16[illegible] 21[illegible] this[illegible]

[illegible] f( )[illegible]

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□23□24□□□□□□
□□□□□□□□□□□25□26□ □□

□28-31□□□□h2□□□□□□□□□□h□□□□□□□□□□□□□□□□□0□

10.3.3 □□□□□□□□

□□C□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□bitcopy□□

□□□□□□□□□□□□□□□□□□□□□□□□C++□□

□□composition□□□

```
//: AUTOCC.CPP -- Automatic copy-constructor
#include <iostream.h>
#include <string.h>

class withCC { // With copy-constructor
public:
  // Explicit default constructor required:
  withCC() {}
  withCC(const withCC&) {
    cout << "withCC(withCC&)" << endl;
  }
};

class woCC { // Without copy-constructor
  enum { bsz = 30 };
  char buf[bsz];
public:
  woCC(const char* msg = 0 ) {
    memset(buf, 0, bsz);
    if(msg) strncpy(buf, msg, bsz);
  }
  void print(const char* msg = 0) const {
    if(msg) cout << msg << ": ";
    cout << buf << endl;
  }
};

class composite {
  withCC WITHCC; // Embedded objects
  woCC WOCC;
public:
  composite() : WOCC("composite()") {}
  void print(const char* msg = 0) {
    WOCC.print(msg);
  }
};

main() {
  composite c;
  c.print("contents of c");
  cout << "calling composite copy-constructor"
       << endl;
  composite c2 = c;  // Calls copy-constructor
  c2.print("contents of c2");
}
```

□ withCC□□□□□□□□□□□□□□□□□□□□□□□□□□□□ composite□ □□□□□□□□□□□□□□ withCC□□□□□□□□□ withCC□□ withCC□□□□□□□□□□□□□□□□□□

□ woCC□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ print( )□□□□□□□□□□□□ composite□□□□□□□□□□□□□□□□□□□□□□□□ 7□□□□□□□□□□□□□ 13□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□ composite□ □□ withCC□□□□□□□□□ woCC□□□□□□□□□□□□□□ WOCC□□□□□□□□□□□□□□□□□□□□□□ composite□□□□□□□□□□□□□□□□□ main( )□□□□□□□□□□□□□□□□□□□□□□□□□

```
composite c2 = c ;
```

□ composite□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□ 13□□□ withCC□□□□□□□□□□□□□□□□□□□□□□□□□□□□ woCC□□□□□□□□□□□□□□□□□□□□□□□□□□□ bitcopy□□□□□□□□□□ composite□□□□□□□□□□□□□□□□□□□□□□□□□□ main□ □□□ composite::print( )□ □□ c2.WOCC□ □□□ c.WOCC□ □□□□□□□□□□□□□□□□□□□□□□□□□□ memberwise initialization□

□□□

10.3.4 □□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

[illegible]

1. [illegible]

[illegible]“ [illegible] ”

[illegible] private[illegible] friend[illegible]

[illegible]

```
//: STOPCC.CPP -- Preventing copy-construction

class noCC {
  int i;
  noCC(const noCC&); // No definition
public:
  noCC(int I = 0) : i(I) {}
};

void f(noCC);

main() {
  noCC n;
//! f(n); // Error: copy-constructor called
//! noCC n2 = n; // Error: c-c called
//! noCC n3(n); // Error: c-c called
}
```

[illegible]

noCC(const noCC&)[illegible]

[illegible] const[illegible]

2. [illegible]

[illegible] iostreams[illegible] get( )[illegible] char&[illegible] get( )[illegible]

```
char c;
cin.get(c);
```

[illegible]

[illegible] const[illegible] const [illegible]

10.4 [illegible]

[illegible] C++[illegible] (pointer-to-member)[illegible] C++[illegible]

[illegible]

```
struct simple { int a ; };
```

[illegible] sp[illegible] so[illegible]

```
sp->a ;
so.a ;
```

[illegible] integer[illegible] ip[illegible] ip[illegible] *[illegible]

```
*ip = 4 ;
```

[illegible] *[illegible] *[illegible] simple[illegible]

```
sp->*pm = 47 ;
so.*pm = 47 ;
```

[illegible] ->*[illegible] .*[illegible] pm[illegible] ‘ *’ [illegible]

```
int simple::*pm ;
```

[illegible]

```
int simple::*pm = &simple::a ;
```

[illegible] simple::a[illegible] “ [illegible] ”[illegible] &simple::a[illegible]

[illegible]

[illegible] pointer-to-member[illegible]

```
int (*fp)(float) ;
```

(*fp)[illegible] int*[illegible]

[illegible]

```
struct simple2 { int f(float); } ;
```

China-pub.com [illegible]

[illegible]

```
int (simple2::*fp) (float) ;
```

[illegible]

```
int (simple2::*fp) (float) = &simple2::f ;
```

[illegible] &[illegible]

```
fp = simple2::f ;
```

- [illegible]

[illegible] (public)[illegible]

```
//: PMEM.CPP -- Pointers to members

class widget {
public:
  void f(int);
  void g(int);
  void h(int);
  void i(int);
};

void widget::h(int) {}

main() {
  widget w;
  widget* wp = &w;
  void (widget::*pmem)(int) = &widget::h;
  (w.*pmem)(1);
  (wp->*pmem)(2);
}
```

[illegible] typedef[illegible] [illegible][1]

```
//: PMEM2.CPP -- Pointers to members
#include <iostream.h>
class widget {
  void f(int) const {cout << "widget::f()\n";}
  void g(int) const {cout << "widget::g()\n";}
  void h(int) const {cout << "widget::h()\n";}
  void i(int) const {cout << "widget::i()\n";}
```

---

[1] [illegible] Owen Mortensen[illegible]

```
    enum { count = 4 };
    void (widget::*fptr[count])(int) const;
  public:
    widget() {
      fptr[0] = &widget::f; // Full spec required
      fptr[1] = &widget::g;
      fptr[2] = &widget::h;
      fptr[3] = &widget::i;
    }
    void select(int I, int J) {
      if(I < 0 || I >= count) return;
      (this->*fptr[I])(J);
    }
    int Count() { return count; }
  };

  main() {
    widget w;
    for(int i = 0; i < w.Count(); i++)
      w.select(i, 47);
  }
```

□□□□□ main( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ Count( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

`fptr[1] = &g ;`

□□□□ g□□

`(this->*fptr[i]) (j);`

□□□□□□□□□□ this□□

10.5 □□

C++□□□□ C□□□□□□□□□□□□□□□□□□□□□□□□ C□□□ C++□□ cast□□□□

C++□□ Algol□ Pascal□□□□□ (reference)□□□

□□

[illegible] (private) [illegible]

[illegible]

10.6 [illegible]

1. [illegible] char& [illegible] main( ) [illegible] char [illegible]

2. [illegible] cout [illegible]

3. [illegible] , [illegible] PASSTRUC.CPP [illegible]

□ 11□　□□□□□

□□□□□□□□□“□□□□”□□□□□□□□□□□□□□□□□□

□□□

□□ C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ operator□□□

11.1 □□□□□

□□□

□□□□□□□□□□□□□□□□□□□□□□□□C□□□□□□□□□□□□“□□□□□□□□□□□□□□C□□□□□□□□□□”□□□

```
1 << 4;
```

□□□□□□□□□□□□□□□□□□□

```
1.414 << 2;
```

□□□□□□□□□□□□□□□□□□□□□□□□□□

11.2 □□

□□□□□□□□□□□□□□□□□□□□□□□□□□□ operator@□□□ @□□□□□□□□□□□□□□□□□□□□□□□□□

1) □□□□□□□□□□□□□□□□□□□□□□□□□

2) □□ —□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□

```
//: OPOVER.CPP -- Operator overloading syntax
#include <iostream.h>

class integer {
  int i;
public:
  integer(int I) { i = I; }
  const integer
```

[illegible]

```
  operator+(const integer& rv) const {
    cout << "operator+" << endl;
    return integer(i + rv.i);
  }
  integer&
  operator+=(const integer& rv){
    cout << "operator+=" << endl;
    i += rv.i;
    return *this;
  }
};

main() {
  cout << "built-in types:" << endl;
  int i = 1, j = 2, k = 3;
  k += i + j;
  cout << "user-defined types:" << endl;
  integer I(1), J(2), K(3);
  K += I + J;
}
```

[illegible]

[illegible]

K += I + J ;

[illegible] +[illegible] ‘ +=’ [illegible] rv[illegible]

11.3 [illegible]

[illegible] C[illegible] C[illegible] **[illegible] ——[illegible]

[illegible] “ [illegible] ” [illegible]

11.3.1 [illegible]

[illegible] integer[illegible] byte[illegible]

```
//: UNARY.CPP -- Overloading unary operators
#include <iostream.h>
class integer {
  long i;
  integer* This() { return this; }
public:
  integer(long I = 0) : i(I) {}
  // No side effects takes const& argument:
  friend const integer&
    operator+(const integer& a);
  friend const integer
    operator-(const integer& a);
  friend const integer
    operator~(const integer& a);
  friend integer*
    operator&(integer& a);
  friend int
    operator!(const integer& a);
  // Side effects don't take const& argument:
  // Prefix:
  friend const integer&
    operator++(integer& a);
  // Postfix:
  friend const integer
    operator++(integer& a, int);
  // Prefix:
  friend const integer&
    operator--(integer& a);
  // Postfix:
  friend const integer
    operator--(integer& a, int);
};

// Global operators:
const integer& operator+(const integer& a) {
  cout << "+integer\n";
  return a; // Unary + has no effect
}
const integer operator-(const integer& a) {
  cout << "-integer\n";
  return integer(-a.i);
}
const integer operator~(const integer& a) {
  cout << "~integer\n";
  return integer(~a.i);
}
```

```
integer* operator&(integer& a) {
  cout << "&integer\n";
  return a.This(); // &a is recursive!
}
int operator!(const integer& a) {
  cout << "!integer\n";
  return !a.i;
}
// Prefix; return incremented value
const integer& operator++(integer& a) {
  cout << "++integer\n";
  a.i++;
  return a;
}
// Postfix; return the value before increment:
const integer operator++(integer& a, int) {
  cout << "integer++\n";
  integer r(a.i);
  a.i++;
  return r;
}
// Prefix; return decremented value
const integer& operator--(integer& a) {
  cout << "--integer\n";
  a.i--;
  return a;
}
// Postfix; return the value before decrement:
const integer operator--(integer& a, int) {
  cout << "integer--\n";
  integer r(a.i);
  a.i--;
  return r;
}

void f(integer a) {
  +a;
  -a;
  ~a;
  integer* ip = &a;
  !a;
  ++a;
  a++;
  --a;
  a--;
}
```

```
// Member operators (implicit "this"):
class byte {
  unsigned char b;
public:
  byte(unsigned char B = 0) : b(B) {}
  // No side effects: const member function:
  const byte& operator+() const {
    cout << "+byte\n";
    return *this;
  }
  const byte operator-() const {
    cout << "-byte\n";
  return byte(-b);
}
const byte operator~() const {
  cout << "~byte\n";
  return byte(~b);
}
byte operator!() const {
  cout << "!byte\n";
  return byte(!b);
}
byte* operator&() {
  cout << "&byte\n";
  return this;
}
// Side effects: non-const member function:
const byte& operator++() { // Prefix
  cout << "++byte\n";
  b++;
  return *this;
}
const byte operator++(int) { // Postfix
  cout << "byte++\n";
  byte before(b);
  b++;
  return before;
}
const byte& operator--() { // Prefix
  cout << "--byte\n";
  --b;
  return *this;
}
const byte operator--(int) { // Postfix
  cout << "byte--\n";
```

```
    byte before(b);
    --b;
    return before;
  }
  };

  void g(byte b) {
    +b;
    -b;
    ~b;
    byte* bp = &b;
    !b;
    ++b;
    b++;
    --b;
    b--;
  }

  main() {
    integer a;
    f(a);
    byte b;
    g(b);
  }
```

□□

• □□□□□

□□□++□ --□□□++a□□□□□□□□□□□ operator++(a);□□□□□□□ a++□□□□□□ operator++(a,int)□□□□□□□□□□□□□□□□□□□□□ □ UNARY.CPP□□□□□□□□□□□□□□ ++b□□□□□□□□ B::perator++( )□□□□□□□□□□□ b++□□□□□□□□ B::operator++(int)□□□□

□□□ int□□□

11.3.2 □□□□□

□□□□□□□□□□□□□□ UNARY.CPP□□□□□□□□□□□□□□□□□□□

```
//: BINARY.CPP -- Overloading binary operators
#include <fstream.h>
#include "..\allege.h"
```

```
ofstream out("binary.out");

class integer { // Combine this with UNARY.CPP
  long i;
public:
  integer(long I = 0) : i(I) {}
  // Operators that create new, modified value:
  friend const integer
    operator+(const integer& left,
              const integer& right);
  friend const integer
    operator-(const integer& left,
              const integer& right);
  friend const integer
    operator*(const integer& left,
              const integer& right);
  friend const integer
    operator/(const integer& left,
              const integer& right);
  friend const integer
    operator%(const integer& left,
              const integer& right);
  friend const integer
    operator^(const integer& left,
              const integer& right);
  friend const integer
    operator&(const integer& left,
              const integer& right);
  friend const integer
    operator|(const integer& left,
              const integer& right);
  friend const integer
    operator<<(const integer& left,
               const integer& right);
  friend const integer
    operator>>(const integer& left,
               const integer& right);
  // Assignments modify & return lvalue:
  friend integer&
    operator+=(integer& left,
               const integer& right);
  friend integer&
    operator-=(integer& left,
               const integer& right);
  friend integer&
    operator*=(integer& left,
```

□□

```
                  const integer& right);
  friend integer&
    operator/=(integer& left,
                  const integer& right);
  friend integer&
    operator%=(integer& left,
                  const integer& right);
  friend integer&
    operator^=(integer& left,
                 const integer& right);
  friend integer&
    operator&=(integer& left,
                 const integer& right);
  friend integer&
    operator|=(integer& left,
                 const integer& right);
  friend integer&
    operator>>=(integer& left,
                  const integer& right);
  friend integer&
    operator<<=(integer& left,
                  const integer& right);
  // Conditional operators return true/false:
  friend int
    operator==(const integer& left,
                 const integer& right);
  friend int
    operator!=(const integer& left,
                 const integer& right);
  friend int
    operator<(const integer& left,
                const integer& right);
  friend int
    operator>(const integer& left,
                const integer& right);
  friend int
    operator<=(const integer& left,
                 const integer& right);
  friend int
    operator>=(const integer& left,
                 const integer& right);
  friend int
    operator&&(const integer& left,
                 const integer& right);
  friend int
    operator||(const integer& left,
                    const integer& right);
  // Write the contents to an ostream:
  void print(ostream& os) const { os << i; }
};

const integer
  operator+(const integer& left,
            const integer& right) {
  return integer(left.i + right.i);
}
const integer
  operator-(const integer& left,
            const integer& right) {
  return integer(left.i - right.i);
}
const integer
  operator*(const integer& left,
            const integer& right) {
  return integer(left.i * right.i);
}
const integer
  operator/(const integer& left,
            const integer& right) {
  allege(right.i != 0, "divide by zero");
  return integer(left.i / right.i);
}
const integer
  operator%(const integer& left,
            const integer& right) {
  allege(right.i != 0, "modulo by zero");
  return integer(left.i % right.i);
}
const integer
  operator^(const integer& left,
            const integer& right) {
  return integer(left.i ^ right.i);
}
const integer
  operator&(const integer& left,
            const integer& right) {
  return integer(left.i & right.i);
}
const integer
  operator|(const integer& left,
            const integer& right) {
```

```
  return integer(left.i | right.i);
}
const integer
  operator<<(const integer& left,
             const integer& right) {
  return integer(left.i << right.i);
}
const integer
  operator>>(const integer& left,
             const integer& right) {
  return integer(left.i >> right.i);
}
// Assignments modify & return lvalue:
integer& operator+=(integer& left,
                    const integer& right) {
   if(&left == &right) {/* self-assignment */}
   left.i += right.i;
   return left;
}
integer& operator-=(integer& left,
                    const integer& right) {
   if(&left == &right) {/* self-assignment */}
   left.i -= right.i;
   return left;
}
integer& operator*=(integer& left,
                    const integer& right) {
   if(&left == &right) {/* self-assignment */}
   left.i *= right.i;
   return left;
}
integer& operator/=(integer& left,
                    const integer& right) {
   allege(right.i != 0, "divide by zero");
   if(&left == &right) {/* self-assignment */}
   left.i /= right.i;
   return left;
}
integer& operator%=(integer& left,
                    const integer& right) {
   allege(right.i != 0, "modulo by zero");
   if(&left == &right) {/* self-assignment */}
   left.i %= right.i;
   return left;
}
integer& operator^=(integer& left,
                    const integer& right) {
   if(&left == &right) {/* self-assignment */}
   left.i ^= right.i;
   return left;
}
integer& operator&=(integer& left,
                    const integer& right) {
   if(&left == &right) {/* self-assignment */}
   left.i &= right.i;
   return left;
}
integer& operator|=(integer& left,
                    const integer& right) {
   if(&left == &right) {/* self-assignment */}
   left.i |= right.i;
   return left;
}
integer& operator>>=(integer& left,
                     const integer& right) {
   if(&left == &right) {/* self-assignment */}
   left.i >>= right.i;
   return left;
}
integer& operator<<=(integer& left,
                     const integer& right) {
   if(&left == &right) {/* self-assignment */}
   left.i <<= right.i;
   return left;
}
// Conditional operators return true/false:
int operator==(const integer& left,
               const integer& right) {
    return left.i == right.i;
}
int operator!=(const integer& left,
               const integer& right) {
    return left.i != right.i;
}
int operator<(const integer& left,
              const integer& right) {
    return left.i < right.i;
}
int operator>(const integer& left,
              const integer& right) {
    return left.i > right.i;
}
```

下载

```
int operator<=(const integer& left,
               const integer& right) {
    return left.i <= right.i;
}
int operator>=(const integer& left,
               const integer& right) {
    return left.i >= right.i;
}
int operator&&(const integer& left,
               const integer& right) {
    return left.i && right.i;
}
int operator||(const integer& left,
               const integer& right) {
    return left.i || right.i;
}

void h(integer& c1, integer& c2) {
  // A complex expression:
  c1 += c1 * c2 + c2 % c1;
  #define TRY(op) \
  out << "c1 = "; c1.print(out); \
  out << ", c2 = "; c2.print(out); \
  out << ";  c1 " #op " c2 produces ";\
  (c1 op c2).print(out); \
  out << endl;
  TRY(+) TRY(-) TRY(*) TRY(/)
  TRY(%) TRY(^) TRY(&) TRY(|)
  TRY(<<) TRY(>>) TRY(+=) TRY(-=)
  TRY(*=) TRY(/=) TRY(%=) TRY(^=)
  TRY(&=) TRY(|=) TRY(>>=) TRY(<<=)
  // Conditionals:
  #define TRYC(op) \
  out << "c1 = "; c1.print(out); \
  out << ", c2 = "; c2.print(out); \
  out << ";  c1 " #op " c2 produces ";\
  out << (c1 op c2); \
  out << endl;
  TRYC(<) TRYC(>) TRYC(==) TRYC(!=) TRYC(<=)
  TRYC(>=) TRYC(&&) TRYC(||)
}

// Member operators (implicit "this"):
class byte { // Combine this with UNARY.CPP
  unsigned char b;
```

```
public:
  byte(unsigned char B = 0) : b(B) {}
  // No side effects: const member function:
  const byte
    operator+(const byte& right) const {
    return byte(b + right.b);
  }
  const byte
    operator-(const byte& right) const {
    return byte(b - right.b);
  }
  const byte
    operator*(const byte& right) const {
    return byte(b * right.b);
  }
  const byte
    operator/(const byte& right) const {
    allege(right.b != 0, "divide by zero");
    return byte(b / right.b);
  }
  const byte
    operator%(const byte& right) const {
    allege(right.b != 0, "modulo by zero");
    return byte(b % right.b);
  }
  const byte
    operator^(const byte& right) const {
    return byte(b ^ right.b);
  }
  const byte
    operator&(const byte& right) const {
    return byte(b & right.b);
  }
  const byte
    operator|(const byte& right) const {
    return byte(b | right.b);
  }
  const byte
    operator<<(const byte& right) const {
    return byte(b << right.b);
  }
  const byte
    operator>>(const byte& right) const {
    return byte(b >> right.b);
  }
```

□ □

```
// Assignments modify & return lvalue.
// operator= can only be a member function:
byte& operator=(const byte& right) {
  // Handle self-assignment:
  if(this == &right) return *this;
  b = right.b;
  return *this;
}
byte& operator+=(const byte& right) {
  if(this == &right) {/* self-assignment */}
  b += right.b;
  return *this;
}
byte& operator-=(const byte& right) {
  if(this == &right) {/* self-assignment */}
  b -= right.b;
  return *this;
}
byte& operator*=(const byte& right) {
  if(this == &right) {/* self-assignment */}
  b *= right.b;
  return *this;
}
byte& operator/=(const byte& right) {
  allege(right.b != 0, "divide by zero");
  if(this == &right) {/* self-assignment */}
  b /= right.b;
  return *this;
}
byte& operator%=(const byte& right) {
  allege(right.b != 0, "modulo by zero");
  if(this == &right) {/* self-assignment */}
  b %= right.b;
  return *this;
}
byte& operator^=(const byte& right) {
  if(this == &right) {/* self-assignment */}
  b ^= right.b;
  return *this;
}
byte& operator&=(const byte& right) {
  if(this == &right) {/* self-assignment */}
  b &= right.b;
  return *this;
}
byte& operator|=(const byte& right) {
    if(this == &right) {/* self-assignment */}
    b |= right.b;
    return *this;
  }
  byte& operator>>=(const byte& right) {
    if(this == &right) {/* self-assignment */}
    b >>= right.b;
    return *this;
  }
  byte& operator<<=(const byte& right) {
    if(this == &right) {/* self-assignment */}
    b <<= right.b;
    return *this;
  }
  // Conditional operators return true/false:
  int operator==(const byte& right) const {
      return b == right.b;
  }
  int operator!=(const byte& right) const {
      return b != right.b;
  }
  int operator<(const byte& right) const {
      return b < right.b;
  }
  int operator>(const byte& right) const {
      return b > right.b;
  }
  int operator<=(const byte& right) const {
      return b <= right.b;
  }
  int operator>=(const byte& right) const {
      return b >= right.b;
  }
  int operator&&(const byte& right) const {
      return b && right.b;
  }
  int operator||(const byte& right) const {
      return b || right.b;
  }
  // Write the contents to an ostream:
  void print(ostream& os) const {
    os << "0x" << hex << int(b) << dec;
  }
};

void k(byte& b1, byte& b2) {
  b1 = b1 * b2 + b2 % b1;

  #define TRY2(op) \
  out << "b1 = "; b1.print(out); \
  out << ", b2 = "; b2.print(out); \
  out << ";  b1 " #op " b2 produces ";\
  (b1 op b2).print(out); \
  out << endl;

  b1 = 9; b2 = 47;
  TRY2(+) TRY2(-) TRY2(*) TRY2(/)
  TRY2(%) TRY2(^) TRY2(&) TRY2(|)
  TRY2(<<) TRY2(>>) TRY2(+=) TRY2(-=)
  TRY2(*=) TRY2(/=) TRY2(%=) TRY2(^=)
  TRY2(&=) TRY2(|=) TRY2(>>=) TRY2(<<=)
  TRY2(=) // Assignment operator

  // Conditionals:
  #define TRYC2(op) \
  out << "b1 = "; b1.print(out); \
  out << ", b2 = "; b2.print(out); \
  out << ";  b1 " #op " b2 produces ";\
  out << (b1 op b2); \
  out << endl;

  b1 = 9; b2 = 47;
  TRYC2(<) TRYC2(>) TRYC2(==) TRYC2(!=) TRYC2(<=)
  TRYC2(>=) TRYC2(&&) TRYC2(||)

  // Chained assignment:
  byte b3 = 92;
  b1 = b2 = b3;
}

main() {
  integer c1(47), c2(9);
  h(c1, c2);
  out << "\n member functions:" << endl;
  byte b1(47), b2(9);
  k(b1, b2);
}
```

□□□□□□□‘ =’ □□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ (self-assignment)□□□□□□□□□□□□□□□□□□□□□□□□‘ +=’ □ A+=A□ □□ A□ □□□□□□□□□□□□□□□□□□□□‘ =’□□□□□□□□□□□□□□□□□□□

[illegible]‘ =’ [illegible]

[illegible] “ [illegible] ”[illegible]

11.3.3 [illegible]

[illegible] UNARY.CPP[illegible] BINARY.CPP[illegible]

1) [illegible] const[illegible] +[illegible] -[illegible] const[illegible] const[illegible] (operator-assignment[illegible] +=[illegible]‘ =’[illegible] (constant)[illegible]

2) [illegible] integer::operator+[illegible] integer[illegible] const[illegible]

3) [illegible] A=B=C[illegible] const[illegible] nonconst[illegible] A=B=C[illegible] nonconst[illegible] A=B[illegible] .foo[illegible] B[illegible] A[illegible] foo( )[illegible] nonconst[illegible]

4) [illegible] int[illegible] bool[illegible] C++[illegible] bool[illegible] int[illegible] typedef[illegible]

5) [illegible] const[illegible] *this[illegible] int [illegible] bool[illegible] (iterator)[illegible] const[illegible] nonconst[illegible] (++A).foo( )[illegible] foo( )[illegible] A[illegible] (A++).foo( )[illegible] foo( )[illegible] ++[illegible] const[illegible] const[illegible]

1. [illegible] const[illegible]

[illegible] const[illegible] +[illegible] f(A+B)[illegible] A+B[illegible] f( ) [illegible] const[illegible] const[illegible]

□□□□□□□□□□□□□ A+B□□□□□□□□□□□□□□□□□□□□□□□□□ (A+B).g( )□□□ g( )□ integer□□□□□□□□□□□□ const□□□□□□□□□□□ const□□□□□□□□□□□ const□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

2. □□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ +□□

```
return integer (left.i + right.i) ;
```

□□□□□□□□□□ “ □□□□□□□□□□ ”□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“ □□□□□□□□□□□□ ”□□□ tmp□□□□□□□□□□□□□□□□□□□□□□□□□□□□ tmp□□□□□□□□□□□□□□□□□□□□□ tmp□□□□□□□□□□□□□□□

```
integer tmp(left.i + right.i) ;
return tmp ;
```

□□□“ □□□□□□ ” □□□

11.3.4 □□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□

□□□□□‘ [ ]’ □□□

□□ , □□□

```
//: COMMA.CPP -- Overloading operator,
#include <iostream.h>

class after {
public:
  const after& operator,(const after&) const {
    cout << "after::operator,()" << endl;
    return *this;
  }
};

class before {};

before& operator,(int, before& b) {
  cout << "before::operator,()" << endl;
  return b;
}

main() {
  after a, b;
  a, b;  // Operator comma called

  before c;
  1, c;  // Operator comma called
}
```

□□□

□□□( )□□

□□□ new□ delete□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□ ->*□□

□□□□□□□□□□□□□□□□□□□□□□□□ (smart)□□□ ->□□□□□□□□□□□□□□□□□□□□□ , □□□□□□□□□□□□ iterator□□

□□□

```
//: SMARTP.CPP -- Smart pointer example
#include <iostream.h>
#include <string.h>

class obj {
  static int i, j;
public:
  void f() { cout << i++ << endl; }
  void g() { cout << j++ << endl; }
};

// Static member definitions:
int obj::i = 47;
int obj::j = 11;

// Container:
class obj_container {
  enum { sz = 100 };
  obj* a[sz];
  int index;
public:
  obj_container() {
    index = 0;
    memset(a, 0, sz * sizeof(obj*));
  }
  void add(obj* OBJ) {
    if(index >= sz) return;
    a[index++] = OBJ;
  }
  friend class sp;
};

// Iterator:
class sp {
  obj_container* oc;
  int index;
public:
  sp(obj_container* OC) {
    index = 0;
    oc = OC;
  }
  // Return value indicates end of list:
  int operator++() { // Prefix
    if(index >= oc->sz) return 0;
    if(oc->a[++index] == 0) return 0;
    return 1;
  }
  int operator++(int) { // Postfix
    return operator++(); // Use prefix version
  }
  obj* operator->() const {
    if(oc->a[index]) return oc->a[index];
    static obj dummy;
    return &dummy;
  }
};

main() {
  const sz = 10;
  obj o[sz];
  obj_container OC;
  for(int i = 0; i < sz; i++)
    OC.add(&o[i]); // Fill it up
    sp SP(&OC); // Create an iterator
    do {
      SP->f(); // Smart pointer calls
      SP->g();
    } while(SP++);
  }
```

□obj□□□□□□□□□□□□□□f()□g()□□□□□□□□□□□□□□□□□□obj_container□□□□add()□□□□□□□□□□□□□□□obj_container□□□□□□obj_container□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□sp□□□friend□□□□□□□□□□□□□□□□□sp□□□□□□□□□□□——□□□□□□□++□□□□□□□□□□□□□□□□--□□□(iterator)□□□□□□□□□□——□□□□□□“□□□□”□□□□□□□□□□□□□□□15□□□□□□

□main()□□□□□□□OC□□obj□□□□□□□SP□□□□□□□□□□□□□□□□□□□

SP->f( ) ; //Smart pointer calls
SP->g( ) ;

□□□□□SP□□□□□□□□□□f( )□g( )□□□□□□□□□□obj*□□□□□□obj*□□□sp::operator->□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□---□□□□□□□□□□□□□□

11.3.5 □□□□□□□□

□□□

□□□□□□□□□□‘.’□□->□□□

□□□□□□□□□□□□‘.*’□□□□□□‘.’□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□Fotran□□□□□□□**□□□□□□□□□□□□□□C□□□□□□□□□C++□□

□□

□□□□□□□□□□□□□□□□□□□□□

11.4 □□□□□□

□□“□□□□□□□□□”□□

□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□□□□□□□□
iostreams□□□□□<<□>>□□□

```
//: IOSOP.CPP -- Iostream operator overloading
// Example of non-member overloaded operators
#include <iostream.h>
#include <strstrea.h>
#include <string.h>
#include "..\allege.h"

class intarray {
  enum { sz = 5 };
  int i[sz];
public:
  intarray() {
    memset(i, 0, sz* sizeof(*i));
  }
  int& operator[](int x) {
    allege(x >= 0 && x < sz,
           "operator[] out of range");
    return i[x];
  }
  friend ostream&
    operator<<(ostream& os,
               const intarray& ia);
  friend istream&
    operator>>(istream& is, intarray& ia);
};

ostream& operator<<(ostream& os,
                    const intarray& ia){
  for(int j = 0; j < ia.sz; j++) {
    os << ia.i[j];
    if(j != ia.sz -1)
      os << ", ";
  }
  os << endl;
  return os;
}

istream& operator>>(istream& is, intarray& ia){
  for(int j = 0; j < ia.sz; j++)
    is >> ia.i[j];
  return is;
}

main() {
  istrstream input("47 34 56 92 103");
  intarray I;
  input >> I;
  I[4] = -1; // Use overloaded operator[]
  cout << I;
}
```

□□□□□□□□□□□ []□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□

```
I[4] = -1 ;
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□ □□□□

```
os << ia.i[j] ;
```

□□□□□□□□□□□□□□□□□□□□□□ IOSTREAM.H□□□□□□□□□□□□□ □□□□□ ostream& operator<<(ostream&,int)□□□□□ ia.i[j]□□□ int□□

□□□□□□□□ istream□ ostream□□□□□□□□□□□□□□□□□□□□□□□ □□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

Murry [1]□□□□□□□□□□□□□□□□□□□□□□

□□□	□□□□
□□□□□□□□□	□□
= ( ) [] ->	□□□□□
+= -= /= *= ^= &= |= %= >>= <<=	□□
□□□□□□□□□	□□□

11.5 □□□□□

□□□□C++□□□□□□□□□□□□□□□□□□‘ =’□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□‘ =’□□□□□□□□□□□□□□□□□□□□□

```
foo B ;
foo A = B ;
A = B ;
```

□2□□□□□□ A□□□□□□□□□□□□□□□□□□□□□□□□□□□□ C++□ □□ □□□□□□□□□□ A□□□□□ foo□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□‘ =’□□□□□□□□□□□□□

---

[1] Rob Murray, C++ Strategies & Tactics, Addision-Wesley, 1993, page 47.

□3□□□□□□□□‘ =’□□□ A□ □ foo::operator=□ □ foo::operator=□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ operator=□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□‘ =’□□□□□□□□□□□□□□□□□□□□□□□□□ =□□□□□□□□□□□□□□□□□□□□□□□

```
//: FEEFI.CPP -- Copying vs. initialization

class fi {
public:
  fi() {}
};

class fee {
public:
  fee(int) {}
  fee(const fi&) {}
};

main() {
  fee f = 1; // fee(int)
  fi FI;
  fee fum = FI; // fee(fi)
}
```

□□□‘ =’□□□‘ =’□

□□□□□□□□‘ =’□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
fee fum(FI) ;
```

□□□□□□□□□□□□□□

□□□‘ =’□□□

□BINARY.CPP□□□□□□□□□‘ =’□□□□□□□□□□□□‘ =’□□□□□□□□□□□□□□□□□□□□□□□□‘ =’□□□□□□□□□□□□□□□□□‘ =’□

```
int operator=(int,foo) ; // global = not allowed!
```

□□□□□□□□□□□□□□□□□□□‘ =’□□□□□□□□□□□□□

□□□□□□□□‘ =’□□□□□□□□□□□□□□□□□□□□□□□□“ □□”□□□□□□□□□□□□□□

```
//: SIMPCOPY.CPP -- Simple operator=()
#include <iostream.h>

class value {
  int a, b;
  float c;
public:
  value(int A = 0, int B = 0, float C = 0.0) {
    a = A;
    b = B;
    c = C;
  }
  value& operator=(const value& rv) {
    a = rv.a;
    b = rv.b;
    c = rv.c;
    return *this;
  }
  friend ostream&
    operator<<(ostream& os, const value& rv) {
      return os << "a = " << rv.a << ", b = "
        << rv.b << ", c = " << rv.c;
    }
};

main() {
  value A, B(1, 2, 3.3);
  cout << "A: " << A << endl;
  cout << "B: " << B << endl;
  A = B;
  cout << "A after assignment: " << A << endl;
}
```

□□□‘ =’ □□

□□(self-assignment)□□

1. □□□□

□□

□□

```
//: COPYMEM.CPP -- Duplicate during assignment
#include <stdlib.h>
#include <string.h>
#include "..\allege.h"
```

```
class withPointer {
  char* p;
  enum { blocksz = 100 };
public:
  withPointer() {
    p = (char*)malloc(blocksz);
    allegemem(p);
    memset(p, 1, blocksz);
  }
  withPointer(const withPointer& wp) {
    p = (char*)malloc(blocksz);
    allegemem(p);
    memcpy(p, wp.p, blocksz);
  }
  withPointer&
  operator=(const withPointer& wp) {
    // Check for self-assignment:
    if(&wp != this)
      memcpy(p, wp.p, blocksz);
    return *this;
  }
  ~withPointer() {
    free(p);
  }
};

main() {}
```

□□□□□□□□□□□□□□□□□□□□□□□ 4□□□□□□□□□□□□□□□□□□□□□□□□□□‘ =’□□□□□□□□□□□□□□□□□□□□□□□‘ =’□□□

□□□□□□□□□□□□□□□□□□□□□□□‘ =’□□□(reference counting)□□ 0□□□□□□□□□

□□(copy-on-write)□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 1□□□

```
//: REFCOUNT.CPP -- Reference count, copy-on-write
#include <string.h>
```

```
#include <assert.h>

class counted {
  class memblock {
    enum { size = 100 };
    char c[size];
    int refcount;
  public:
    memblock() {
      memset(c, 1, size);
      refcount = 1;
    }
    memblock(const memblock& rv) {
      memcpy(c, rv.c, size);
      refcount = 1;
    }
    void attach() { ++refcount; }
    void detach() {
      assert(refcount != 0);
      // Destroy object if no one is using it:
      if(--refcount == 0) delete this;
    }
    int count() const { return refcount; }
    void set(char x) { memset(c, x, size); }
    // Conditionally copy this memblock.
    // Call before modifying the block; assign
    // resulting pointer to your block;
    memblock* unalias() {
      // Don't duplicate if not aliased:
      if(refcount == 1) return this;
      --refcount;
      // Use copy-constructor to duplicate:
      return new memblock(*this);
    }
  } * block;
public:
  counted() {
    block = new memblock; // Sneak preview
  }
  counted(const counted& rv) {
    block = rv.block; // Pointer assignment
    block->attach();
  }
  void unalias() { block = block->unalias(); }
  counted& operator=(const counted& rv) {
      // Check for self-assignment:
      if(&rv == this) return *this;
      // Clean up what you're using first:
      block->detach();
      block = rv.block; // Like copy-constructor
      block->attach();
      return *this;
    }
    // Decrement refcount, conditionally destroy
    ~counted() { block->detach(); }
    // Copy-on-write:
    void write(char value) {
      // Do this before any write operation:
      unalias();
      // It's safe to write now.
      block->set(value);
    }
  };

  main() {
    counted A, B;
    counted C(A);
    B = A;
    C = C;
    C.write('x');
  }
```

□ □ □ memblock□ □ □ □ □ □ □ □ □ □ □ □□ □ □ □ □ □ block□ memblock□

□ □ attach( )□ □ □ □ □ memblock□ detach( )□ □ □ □ □ □ □ □ □ □ □ □ □ □ 0□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ delete this □ □ □ □ □ □ □ □ □ □ □ □ □

□ □ □ □ □ set( )□ memblock□ □ □ □ □ □ □ □ □ □ conted::unalias( )□ conted::unalias( )□ □ memblock::unalias( )□ □ □ □ □ □ □ □ □ □ □ □ □ 1□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □□ □ □ □ □ □ □ □ □ □ block□ □ □ □ □ □ □ □ □ □ □ □ □ □ 1□ □ □ □ □ □ □ □ □ □ □ □

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ C++□ □ □ new□ delete□ □ C□ □ □ malloc( )□ free( )□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ new□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ delete□ new □ delete□ malloc( )□ free( )□ □ □

□ □ □ □ □ □ □ □ □ □ block□ □ block□ memblock::attach( )□ □ □ □ □ □ □

□ □ □ ‘ =’ □ □ ‘ =’ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ memblock□ □ detach( )□ memblock□ □ □ □ □ □ □ □ □

‘ =’ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□ detach( )□□□□□□ memblock□

□□□“ □□□□□□□□□□□□□□□ ” □□□□□□ write( )□□□□□□□□□□□□□□□□□□□□□□□□ unalias( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□ main( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□‘ =’ □□□□□□□ main( )□□□□□□□ C□□ write( )□□□□□□□□□ C□□□□□□□ A□□□□□

2. □□□□

□□ REFCOUNT.CPP□□□□□□□□

```
//: RCTRACE.CPP -- REFCOUNT.CPP w/ trace info
#include <string.h>
#include <fstream.h>
#include <assert.h>
ofstream out("rctrace.out");

class counted {
  class memblock {
    enum { size = 100 };
    char c[size];
    int refcount;
    static int blockcount;
    int blocknum;
  public:
    memblock() {
      memset(c, 1, size);
      refcount = 1;
      blocknum = blockcount++;
    }
    memblock(const memblock& rv) {
      memcpy(c, rv.c, size);
      refcount = 1;
      blocknum = blockcount++;
      print("copied block");
      out << endl;
      rv.print("from block");
    }
    ~memblock() {
    out << "\tdestroying block "
        << blocknum << endl;
   }
   void print(const char* msg = "") const {
      if(*msg) out << msg << ", ";
      out << "blocknum:" << blocknum;
      out << ", refcount:" << refcount;
    }
    void attach() { ++refcount; }
    void detach() {
      assert(refcount != 0);
      // Destroy object if no one is using it:
      if(--refcount == 0) delete this;
    }
    int count() const { return refcount; }
    void set(char x) { memset(c, x, size); }
    // Conditionally copy this memblock.
    // Call before modifying the block; assign
    // resulting pointer to your block;
    memblock* unalias() {
      // Don't duplicate if not aliased:
      if(refcount == 1) return this;
      --refcount;
      // Use copy-constructor to duplicate:
      return new memblock(*this);
    }
  } * block;
  enum { sz = 30 };
  char id[sz];
public:
  counted(const char* ID = "tmp") {
    block = new memblock; // Sneak preview
    strncpy(id, ID, sz);
  }
  counted(const counted& rv) {
    block = rv.block; // Pointer assignment
    block->attach();
    strncpy(id, rv.id, sz);
    strncat(id, " copy", sz - strlen(id));
  }
  void unalias() { block = block->unalias(); }
  void addname(const char* nm) {
    strncat(id, nm, sz - strlen(id));
  }
  counted& operator=(const counted& rv) {
    print("inside operator=\n\t");
    if(&rv == this) {
      out << "self-assignment" << endl;
    return *this;
  }
  // Clean up what you're using first:
  block->detach();
  block = rv.block; // Like copy-constructor
  block->attach();
  return *this;
}
// Decrement refcount, conditionally destroy
~counted() {
  out << "preparing to destroy: " << id
      << endl << "\tdecrementing refcount ";
  block->print();
  out << endl;
  block->detach();
}
// Copy-on-write:
void write(char value) {
  unalias();
  block->set(value);
}
void print(const char* msg = "") {
  if(*msg) out << msg << " ";
  out << "object " << id << ": ";
  block->print();
  out << endl;
  }
};

int counted::memblock::blockcount = 0;

main() {
  counted A("A"), B("B");
  counted C(A);
  C.addname(" (C) ");
  A.print();
  B.print();
  C.print();
  B = A;
  A.print("after assignment\n\t");
  B.print();
  out << "Assigning C = C" << endl;
  C = C;
  C.print("calling C.write('x')\n\t");
  C.write('x');
  out << endl << "exiting main()" << endl;
}
```

□□ memblock□ □□□ static□ □□□ blockcount□ □□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□ blocknum□ □□□□□□□□□□□□ □□□□□□□□□□ print( )□□□□□□□□□□□□

□ counted□ □□□□□□ id□ □□□□□□□□ counted□ □□□□□□□□□□ memblock□ □□□□□□□□ block□ □□□□□□□□□□□ memblock□ □□□□□□□□ □□□□□□□□□□□□ “ copy” □□□□□□□□□□□□□□□□ addname( )□□□□ □ id□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□

□□□□□□□□

```
object A: blocknum:0, refcount:2
object B: blocknum:1, refcount:1
object A copy (C) : blocknum:0, refcount:2
inside operator=
       object B: blocknum:1, refcount:1
      destroying block 1
after assignment
       object A: blocknum:0, refcount:3
object B: blocknum:0, refcount:3
Assigning C = C
inside operator=
       object A copy (C) : blocknum:0, refcount:3
self-assignment
calling C.write('x')
       object A copy (C) : blocknum:0, refcount:3
copied block, blocknum:2, refcount:1
from block, blocknum:0, refcount:2
exiting main()
preparing to destroy: A copy (C)
      decrementing refcount blocknum:2, refcount:1
      destroying block 2
preparing to destroy: B
      decrementing refcount blocknum:0, refcount:2
preparing to destroy: A
      decrementing refcount blocknum:0, refcount:1
      destroying block 0
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

3. □□□□□□□‘ =’

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ type::operator=(type)□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□‘ =’ □□□□□□ □□□□□□□□ (memberwise assignment)□□□□□□□

```
//: AUTOEQ.CPP -- Automatic operator=()
#include <iostream.h>
```

```
class bar {
public:
  bar& operator=(const bar&) {
    cout << "inside bar::operator=()" << endl;
    return *this;
  }
};

class foo {
  bar B;
};

main() {
  foo a, b;
  a = b; // Prints: "inside bar::operator=()"
}
```

□ foo□□□□□□□□‘ =’□□ bar::operator=□

□□‘ =’□□□□□□□□□□□□□□□□□□□□□□‘ =’□□□private□□□□□□□□□□□□□□□□□□□□□□

11.6 □□□□□□

□C□ C++□□□ C++□□□

11.6.1 □□□□□□

□□□

```
//: AUTOCNST.CPP -- Type conversion constructor

class one {
public:
  one() {}
};

class two {
public:
  two(const one&) {}
};

void f(two) {}
```

```
main() {
  one One;
  f(One); // Wants a two, has a one
}
```

[illegible] f( )[illegible] one[illegible] f( )[illegible] two [illegible] one [illegible] two[illegible] two::two(one)[illegible] two::two(one)[illegible] two[illegible] f( )[illegible]

[illegible] f( )[illegible] two[illegible] f( )[illegible]

- [illegible]

[illegible] explicit[1][illegible] two[illegible]

```
class one {
public:
  one() {}
};

class two {
public:
  explicit two(const one&) {}
};

void f(two) {}

main() {
  one One;
//!  f(One); // no auto conversion allowed
  f(two(One)); // OK -- user performs conversion
}
```

[illegible] two[illegible] f(two(One))[illegible] One[illegible] two[illegible]

11.6.2 [illegible]

[illegible] operator[illegible] ——[illegible]

---

[1] [illegible] explicit[illegible]

```
//: OPCONV.CPP -- Op overloading conversion

class three {
  int i;
public:
  three(int I = 0, int = 0) : i(I) {}
};

class four {
  int x;
public:
  four(int X) : x(X) {}
  operator three() const { return three(x); }
};

void g(three) {}
main() {
  four Four(1);
  g(Four);
  g(1);  // Calls three(1,0)
}
```

[illegible]

- [illegible]

[illegible]

```
//: REFLEX.CPP -- Reflexivity in overloading

class number {
  int i;
public:
  number(int I = 0) { i = I; }
  const number
  operator+(const number& n) const {
    return number(i + n.i);
  }
  friend const number
    operator-(const number&, const number&);
};
```

```
const number
  operator-(const number& n1,
            const number& n2) {
    return number(n1.i - n2.i);
}

main() {
  number a(47), b(11);
  a + b; // OK
  a + 1; // 2nd arg converted to number
//! 1 + a; // Wrong! 1st arg not of type number
  a - b; // OK
  a - 1; // 2nd arg converted to number
  1 - a; // 1st arg converted to number
}
```

[illegible] number[illegible] +[illegible] (friend)[illegible] -[illegible] int[illegible] int[illegible] number[illegible] main( )[illegible] number[illegible] number[illegible] number[illegible] +[illegible] int[illegible] number::operator+[illegible] int[illegible] number[illegible] int[illegible] +[illegible] number[illegible] number::operator+[illegible] number[illegible]

[illegible] -[illegible] number[illegible] 1-a[illegible] number[illegible]

[illegible] 1-1[illegible] number[illegible] -[illegible] C[illegible] “[illegible]”[illegible] 1-1[illegible]

11.6.3 [illegible] strings

[illegible] string[illegible] C[illegible]

```
//: STRINGS1.CPP -- No auto type conversion
#include <string.h>
#include <stdlib.h>
#include "..\allege.h"

class string {
  char* s;
public:
  string(const char* S = "") {
    s = (char*)malloc(strlen(S) + 1);
    allegemem(s);
    strcpy(s, S);
  }
  ~string() { free(s); }
  int Strcmp(const string& S) const {
    return ::strcmp(s, S.s);
  }
  // ... etc., for every function in string.h
};

main() {
  string s1("hello"), s2("there");
  s1.Strcmp(s2);
}
```

□□□□□ strcmp( )□□□□□□□ STRING.H□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ STRING.H□□□□□□□□□□□□□

```
//: STRINGS2.CPP -- With auto type conversion
#include <string.h>
#include <stdlib.h>
#include "..\allege.h"

class string {
  char* s;
public:
  string(const char* S = "") {
    s = (char*)malloc(strlen(S) + 1);
    allegemem(s);
    strcpy(s, S);
  }
  ~string() { free(s); }
  operator const char*() const { return s; }
};

main() {
  string s1("hello"), s2("there");
  strcmp(s1, s2); // Standard C function
  strspn(s1, s2); // Any string function!
}
```

□□□□□□□□□□ string□□□ char*□□□□□□□□□□□ char*□□□□□□□□□□ string□□□

11.6.4 □□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□X□□□operator Y( ) □□□□□□□□Y□□□□□□□□□□□□□□□□□□□□□Y□□□□□□□□□X□□□□□□□□□□□□□□□□□□□□□□□□□□X□Y□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
//: AMBIG.CPP -- Ambiguity in type conversion

class Y; // Class declaration

class X {
public:
  operator Y() const; // Convert X to Y
};

class Y {
public:
  Y(X); // Convert X to Y
};

void f(Y);

main() {
  X x;
//! f(x); // Error: ambiguous conversion
}
```

□□□(fan-out)□

```
//: FANOUT.CPP -- Type conversion fanout

class A {};
class B {};

class C {
public:
  operator A() const;
  operator B() const;
};

// Overloaded h():
void h(A);
void h(B);

main() {
  C c;
//! h(c); // Error: C -> A or C -> B ???
}
```

[illegible] C[illegible] A[illegible] B[illegible] h( )[illegible] main( )[illegible]

[illegible] make_A( )[illegible] make_B( )[illegible]

- [illegible]

[illegible] 11.5[illegible] FEEFI.CPP[illegible]

```
//: FEEFI2.CPP -- Copying vs. initialization

class fi {};

class fee {
public:
  fee(int) {}
  fee(const fi&) {}
};

class fo {
  int i;
public:
  fo(int x = 0) { i = x; }
  operator fee() const { return fee(i); }
};

main() {
  fo FO;
  fee fiddle = FO;
}
```

[illegible] fo[illegible] fee fiddle[illegible] fo[illegible] fee[illegible] fee[illegible] fee[illegible] '=' [illegible]

```
fee fiddle = FO ;
```

[illegible]

11.7 [illegible]

[illegible]

11.8 [illegible]

1. [illegible] ++[illegible]

2. [illegible] private char[illegible] iostream[illegible] <<[illegible] >>[illegible] IOSOP.CPP [illegible] fstreams[illegible] strstreams[illegible] stdiostreams(cin[illegible] cout)[illegible]

3. [illegible] +[illegible] -[illegible] *[illegible] /[illegible] number[illegible] int( )[illegible]

4. [illegible] UNARY.CPP[illegible] BINARY.CPP[illegible]

5. [illegible] FANOUT.CPP[illegible]

[illegible] 12[illegible] [illegible]

[illegible] CAD[illegible]

[illegible] C[illegible] malloc( )[illegible] free( )[illegible] malloc( )[illegible]

[illegible] C++[illegible]

1) [illegible] C++[illegible]

2) [illegible]

3) [illegible]

[illegible]

C++[illegible]

[illegible] malloc( )[illegible] free( )[illegible]

[illegible] C++[illegible] new[illegible] delete[illegible]

12.1 [illegible]

[illegible] C++[illegible]

1) [illegible]

2) [illegible]

[illegible] 2[illegible] C++[illegible] ——[illegible]

[illegible] 1[illegible]

1) [illegible]

2) [illegible]

3) [illegible]

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□

□□

12.1.1 C□□□□□□□□□□□

□□□□□□□□□□□□□ C□□□□□□□□□□□□□□□□□□□□□□□□□□□□ malloc( )□□□□□□ calloc( )□ realloc( )□□□□□□□□□□□□ free( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ C□□□□□□□□□□□□□□□□□□□□□□□

```
//: MALCLASS.CPP -- Malloc with class objects
// What you'd have to do if not for "new"
#include <stdlib.h> // Malloc() & free()
#include <string.h> // Memset()
#include "..\allege.h"
#include <iostream.h>

class obj {
  int i, j, k;
  enum { sz = 100 };
  char buf[sz];
public:
  void initialize() { // Can't use constructor
    cout << "initializing obj" << endl;
    i = j = k = 0;
    memset(buf, 0, sz);
  }
  void destroy() { // Can't use destructor
    cout << "destroying obj" << endl;
  }
};

main() {
  obj* Obj = (obj*)malloc(sizeof(obj));
  allegemem(Obj);
  Obj->initialize();
  // ... sometime later:
  Obj->destroy();
  free(Obj);
}
```

□□□□□□□□□□□□□□□□□ malloc( )□□□□□□□□

China-pub.com [illegible]

```
obj* Obj = (obj*)malloc(sizeof(obj)) ;
```

[illegible] malloc( )[illegible] void*[illegible] C++[illegible] void* [illegible]

[illegible] malloc( )[illegible] 0[illegible]

[illegible]

```
Obj->initialize( ) ;
```

[illegible] ——[illegible]

[illegible] C[illegible] C[illegible] C++[illegible] C[illegible]

12.1.2 [illegible] new

C++[illegible] new[illegible] new[illegible] new[illegible]

```
foo *fp = new foo(1,2) ;
```

[illegible] malloc(sizeof(foo))[illegible] 1[illegible] 2[illegible] foo[illegible] this[illegible] fp[illegible] ——[illegible] foo[illegible]

[illegible] new[illegible]

[illegible] new[illegible] new[illegible]

```
foo *fp = new foo ;
```

[illegible] ——[illegible] ,[illegible]

12.1.3 [illegible] delete

new[illegible] delete[illegible] delete[illegible] free( )[illegible] new[illegible] delete[illegible]

```
delete fp ;
```

[illegible] foo[illegible]

delete[illegible] new[illegible] malloc( )[illegible] calloc( )[illegible] realloc( )[illegible] delete[illegible] new[illegible] delete[illegible]

□□malloc( )□free( )□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□0□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□0□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

12.1.4 □□□□□□□

□□□□□□□□□□□□□□□□

```
//: NEWDEL.CPP -- Simple demo of new & delete
#include <iostream.h>

class tree {
  int height;
public:
  tree(int Height) {
    height = Height;
  }
  ~tree() { cout << "*"; }
  friend ostream&
  operator<<(ostream& os, const tree* t) {
    return os << "tree height is: "
              << t->height << endl;
  }
};
main() {
  tree* T = new tree(40);
  cout << T;
  delete T;
}
```

□□□□□□tree□□□□□□□□□□□□□□□□□□□□□□□□ <<□□□ostream□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□friend□□□□□□□□□□□□□□□□□□□□□□□□□□——□□ostream&□□□□□□□□□□□□□□□□ostream&□□

12.1.5 □□□□□□□

□□calloc( )□realloc( )□□malloc( )□

1) □□malloc( )□□□□□□□□□□□□□□

2) □□□——□□□□□malloc( )□□□□□□□□□

3) □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□malloc( )□□□□□□□□□□□□□□□□free( )□□□□□□□□□□□□□□□□□

[illegible] malloc( )[illegible]

12.2 [illegible]

[illegible] new[illegible] delete[illegible] stash[illegible] stack[illegible]

12.2.1 [illegible] string[illegible]

[illegible] stash[illegible] stack[illegible] “ [illegible] ” [illegible] stash[illegible] stack[illegible] delete[illegible] void[illegible] delete[illegible] void[illegible] stash[illegible] stack[illegible] 13[illegible] 15[illegible]

[illegible] delete[illegible] String[illegible] String[illegible]

```
//: STRINGS.H -- Simple string class
// Can only be built on the heap
#ifndef STRINGS_H_
#define STRINGS_H_
#include <string.h>
#include <iostream.h>

class String {
  char* s;
  String(const char* S) {
    s = new char[strlen(S) + 1];
    strcpy(s, S);
  }
  // Prevent copying:
  String(const String&);
  void operator=(String&);
public:
  // Only make Strings on the heap:
  friend String* makeString(const char* S) {
    return new String(S);
  }
  // Alternate approach:
  static String* make(const char* S) {
     return new String(S);
   }
   ~String() { delete s; }
   operator char*() const { return s;}
    char* str() const { return s; }
    friend ostream&
      operator<<(ostream& os, const String& S) {
        return os << S.s;
    }
  };
  #endif // STRINGS_H_
```

□ private□ private□ ‘ =’ □ String□ □ □ □ □ □ □ □ □ □ String□ □

□ □ □ □ □ □ □ □ □ □ □ □□ □ □ □ □ □□ □ □ □ □ □ □friend□ □□ □ □ makeString( )□□ □ □ □ □ □ “ □ □ ” □ □ □ □ □ □ □ □ □ □ □ □ static□ □ □ □ □ □ □ make( )□ □ □ □ String::make( )□ □

□ □ □ □ □ □ □ □ □ □ □ □ □

```
s = new char[strlen(S) +1] ;
```

□ char□ □ □□ □

□ char*□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ char*□ □ □ □ □ □ □ □ □ □ String□ □ □ □ □ □ □ □ iostream□ □ □ □ □ □ □ □ iostream□ □ □ □ □ □ □ □ □ String□ □ □

12.2.2 stash□ □

□ □ 5□ □ □ □ □ stash□ □ □ □ □ □ □ □ □ □ □ □ □ 5□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ pstash□ 5□ □ □ □ □ □ □ □ □ □ □ stash□ □ □ □ □ □ stash□ □ □ □ □ □ □ □ □ □ □ □ new□ delete□ □

□ □ □ □ □ “ pointer stash” □ □ □ □ □

```
//: PSTASH.H -- Holds pointers instead of objects
#ifndef PSTASH_H_
#define PSTASH_H_

class pstash {
  int quantity; // Number of storage spaces
  int next; // Next empty space
   // Pointer storage:
  void** storage;
  void inflate(int increase);
public:
  pstash() {
    quantity = 0;
    storage = 0;
    next = 0;
  }
  // No ownership:
  ~pstash() { delete storage; }
  int add(void* element);
  void* operator[](int index) const; // Fetch
  // Number of elements in stash:
  int count() const { return next; }
};
#endif // PSTASH_H_
```

□□□□□□□□□□□□□□□□□ storage□□□ void□□□□□□□□ new□□ malloc( )□□□□□□□□□□□□□□□□□□□□□□□□□ void*□□□□□□□□□□□□□□□□ void□□□□□□

storage = new void*[quantity = Quantity] ;

□□□□□□ void□□□ void□□□□□□□□□□

□□□□□□□□□□□□ []□□□□□ fetch( )□□□□□□□□□□□□□□□□□□□□□□ void□□□

□□□□□□□□□□□□□□

```
//: PSTASH.CPP -- Pointer stash definitions
#include "..\11\pstash.h"
#include <iostream.h>
#include <string.h> // Mem functions

int pstash::add(void* element) {
  const InflateSize = 10;
  if(next >= quantity)
    inflate(InflateSize);
  storage[next++] = element;
  return(next - 1); // Index number
}

// Operator overloading replacement for fetch
void* pstash::operator[](int index) const {
  if(index >= next || index < 0)
    return 0;  // Out of bounds
  // Produce pointer to desired element:
  return storage[index];
}
```

□ □

```
void pstash::inflate(int increase) {
  const psz = sizeof(void*);
  // realloc() is cleaner than this:
  void** st = new void*[quantity + increase];
  memset(st, 0, (quantity + increase) * psz);
  memcpy(st, storage, quantity * psz);
  quantity += increase;
  delete storage; // Old storage
  storage = st; // Point to new memory
}
```

□□□□□□□□□□□□□□□□□□□□ add( )□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□

□□□□□□ inflate( )□□□□□□□□□□□□□□□□□□□□□□□□□□□ realloc( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ new□ realloc□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□ malloc( )□ realloc( )□ free( )□ new□ delete□□□□□□□□□□□□□ □□□ malloc( )□□□□□□□ new( )□ delete( )□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□

- □□□□□□

□□□□□□□ pstash□ □ stash□ □□□□□□□□□□□

```
//: PSTEST.CPP -- Test of pointer stash
#include "..\11\pstash.h"
#include "..\11\strings.h"
#include <fstream.h>
#include "..\allege.h"

main() {
  pstash intStash;
  // new works with built-in types, too:
  for(int i = 0; i < 25; i++)
    intStash.add(new int(i)); // Pseudo-constr.
  for(int u = 0; u < intStash.count(); u++)
    cout << "intStash[" << u << "] = "
         << *(int*)intStash[u] << endl;

  ifstream infile("pstest.cpp");
  allegefile(infile);
  const bufsize = 80;
  char buf[bufsize];
  pstash stringStash;
  // Use global function makeString:
  for(int j = 0; j < 10; j++)
```

```
    if(infile.getline(buf, bufsize))
      stringStash.add(makeString(buf));
  // Use static member make:
  while(infile.getline(buf, bufsize))
    stringStash.add(String::make(buf));
  // Print out the strings:
  for(int v = 0; stringStash[v]; v++) {
    char* p = *(String*)stringStash[v];
    cout << "stringStash[" << v << "] = "
         << p << endl;
  }
}
```

□□□□□□ stash□□□□□□□□□□□□□□□□□□□□ new□□□□□□□□□□□□□□□□□

intStash.add(new int(i)) ;

□□□□□ new int(i) □□□□□□□□□□□□□□□□□ int□□□□□□□□□□□□□ int□□□□□□□□ i□

□□□□□□□□□□ pstash::operator[]□□□□□□□□□□□□□□□□□□□□□□□□□□□ pstash□□□□□□□□□□□□□□□ void□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□ 2□□□□□□□□□□□□□□□□□□□ psatsh□□□□□□□□□□ String□□□□□□□□□□ makeString( )□ String::make( )□□□□□□□□□□□□□□□□ (static)□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□

char* p = *(String*)stringStash[i] ;

□□□□ []□□□□□□□□□□□□□ String*□□□□□□□□□□□□□ String*□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ char*□□□□□□□ String□□□□□□□ String□□□□□□□□□□□□□□□□□ char*□

□□

12.2.3 stack□ □

stack□□□□□□□□ 4□□□□□□□□□□□□□□□□□□□

```
//: STACK11.H -- New version of stack
#ifndef STACK11_H_
#define STACK11_H_

class stack {
  struct link {
    void* data;
    link* next;
    link(void* Data, link* Next) {
        data = Data;
        next = Next;
      }
    } * head;
  public:
    stack() { head = 0; }
    ~stack();
    void push(void* Data) {
      head = new link(Data,head);
    }
    void* peek() const { return head->data; }
    void* pop();
  };
  #endif // STACK11_H_
```

□□□struct link□□□□□□□□□□□□□□□□stack::push( )□□new□□link::link□□□□□□□□□□□data□next□□□□□□stack::push( )□□□□□□□□□□□□link□□□□□□□□□□□link□□□□

head = new link(Data, head) ;

□□□□□□□□□□□□□4□□□□□□□□□□□□□□□□□□□□□□□□

```
//: STACK11.CPP -- New version of stack
#include <stdlib.h>
#include "..\11\stack11.h"

void* stack::pop() {
  if(head == 0) return 0;
  void* result = head->data;
  link* oldHead = head;
  head = head->next;
  delete oldHead;
  return result;
}

stack::~stack() {
  link* cursor = head;
  while(head) {
    cursor = cursor->next;
    delete head;
    head = cursor;
  }
}
```

□□□□□□□□□□□□□□□delete□□free( )□

□stash□□□□□void□□□□□□□□□□□□□□□stack□□□□□□□□□□

stack□□□

```
//: STKTST11.CPP -- Test new stack
#include "..\11\stack11.h"
#include "..\11\strings.h"
#include <fstream.h>
#include "..\allege.h"

main() {
  // Could also use command-line argument:
  ifstream file("stktst11.cpp");
  allegefile(file);
  const bufsize = 100;
  char buf[bufsize];
  stack textlines;
  // Read file and store lines in the stack:
  while(file.getline(buf,bufsize))
    textlines.push(String::make(buf));
  // Pop lines from the stack and print them:
  String* s;
  while((s = (String*)textlines.pop()) != 0)
    cout << *s << endl;
}
```

□ stash□□□□□□□□□□□□□□□□□□□□□□□□□ String□□□□□□ stack□□□□□□□□□□□□□□□□□ stack□□□□□ stack□□□□□□□□□□□□□□□□□

12.3 □□□□□ new□ delete

□ C++□□□ 4□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□ new□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
foo* fp = new foo[100] ;
```

□□□□□□ 100□ foo□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ foo*□□□□□□□□□□□□□□□□□□□□□□□□□

```
foo* fp2 = new foo ;
```

□□□□□□□□□□□□□□□□□□□ fp□□□□□□□□□□□□□□□□□ fp[2]□□□

```
delete fp2 ;  //OK
delete fp ;   // Not the desired effect
```

□□□□□□□□□□□□□□□□□ foo□□□□□□□□□□□□□□□□□□ fp2□□□□□□□□□□□ fp□□□ 99□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

delete []fp ;

□□

delete [100]fp ;

□□□

□□□□□□□

□□□□□□□□□□□ fp□□□

int const* q = new int[10] ;

□

const int* q = new int[10] ;

□□□□□□□□□□□ const□ □ int □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

int* const q = new int[10] ;

□□□ q□ □□□□□□□□□□□□□ q□ □□□□□□□ q++□ □□□□□□□□□□□□□□□□□□□□

12.4 □□□□

□□□□ new□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□ new-handler□ □□□□□□□□□□□□□□□□□□□□□□□□ 0□ □□□□□□□□□□□□

□□ new-handler□ □□□□□□□□□□□ (throw an exception)□ □□□□□□ 17□ □□□□□□□□□□□□□□□□□□□□□□□□ “ □□□□□ ” □□□□□ new-handler□ □□

□□□□ NEW.H□ □□□□□□□□□□□□□□□□□□ set_new_handler( )□ □□□□□□□□ new-handler□

```
//: NEWHANDL.CPP -- Changing the new-handler
#include <iostream.h>
#include <stdlib.h>
#include <new.h>

void out_of_memory() {
  cerr << "memory exhausted!" << endl;
  exit(1);
}

main() {
  set_new_handler(out_of_memory);
  while(1)
    new int[1000]; // Exhausts memory
}
```

new-handler [illegible] void[illegible] while[illegible] int[illegible] new[illegible] new-handler[illegible]

[illegible] new-handler[illegible]

12.5 [illegible] new[illegible] delete

[illegible] new[illegible] new[illegible] delete[illegible] delete[illegible] new[illegible] delete[illegible]

[illegible] new[illegible] delete[illegible] C++[illegible] new[illegible] delete[illegible]

[illegible]

[illegible] new[illegible] delete[illegible] C++[illegible]

[illegible] new[illegible] delete[illegible] new[illegible] new [illegible] new[illegible] delete [illegible]

[illegible] new[illegible] new[illegible] 0[illegible] new-handler[illegible] bad_alloc[illegible] 17[illegible]

[illegible] new[illegible] delete[illegible]

12.5.1 [illegible] new[illegible] delete

[illegible] new[illegible] delete[illegible]

□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□

□□□new□□□□□size_t□□0□□□□□□□□□□□□□□□□□□□0□□□□□□new-handler□□□□□□□□□□□□□□□□

□□□new□□□□□□□ void*□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□——□□□

□□□ delete□□□□□□□□□□□ new□□□□□□ void*□□□□□ void*□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ delete□□□□□□ void□

□□□□□□□□□□□□□□ new□ delete□□□□□□□□

```
//: GLOBLNEW.CPP -- Overload global new/delete
#include <stdio.h>
#include <stdlib.h>

void* operator new(size_t sz) {
  printf("operator new: %d bytes\n", sz);
  void* m = malloc(sz);
  if(!m) puts("out of memory");
  return m;
}

void operator delete(void* m) {
  puts("operator delete");
  free(m);
}

class s {
  int i[100];
public:
  s() { puts("s::s()"); }
  ~s() { puts("s::~s()"); }
};

main() {
  puts("creating & destroying an int");
  int* p = new int(47);
  delete p;
  puts("creating & destroying an s");
  s* S = new s;
  delete S;
  puts("creating & destroying s[3]");
  s* SA = new s[3];
  delete []SA;
}
```

□□□□□□□□ new□ delete□□□□□□□□□□□□□□□□□□□□ C□□□ malloc( )□ free( )□□□□□□ new□ delete□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ printf( )□ puts( )□□□ iostreams□□□□□□ iostream□□□□□□□□ cin□ cout□ cerr□□□□□□ new□□□□□□ printf( )□□□□□□□□□□□□□□□□ new□□□□□□□

□ main( )□□□□□□□□□□□□□□□□□□□□□□□□□ new□ delete□□□□□□□□□□□□□□ s□□ new□ delete□

12.5.2 □□□□□□ new□ delete

□□□□□□ new□ delete□□□□□□ static□□□□□□□ static□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ new□□□□□□□□□□□□□□□□□ new□□□□□□□ new□□□□□□ new□ delete□□□□□□□□□□□□□□□□□□□ new□ delete□□

□□□□□□□□□□□ framis□□ framis□□ (bytes)□□□□□□□ (byte)□□□□□□□□□

```
//: FRAMIS.CPP -- Local overloaded new & delete
#include <stddef.h> // Size_t
#include <fstream.h>
ofstream out("framis.out");

class framis {
  char c[10];
  static unsigned char pool[];
  static unsigned char alloc_map[]; // Alloc map
public:
  enum { psize = 100 };  // # of frami allowed
  framis() { out << "framis()\n"; }
  ~framis() { out << "~framis() ... "; }
  void* operator new(size_t);
  void operator delete(void*);
};
unsigned char framis::pool[psize * sizeof(framis)];
unsigned char framis::alloc_map[psize] = {0};

// Size is ignored -- assume a framis object
void* framis::operator new(size_t) {
  for(int i = 0; i < psize; i++)
    if(!alloc_map[i]) {
      out << "using block " << i << " ... ";
      alloc_map[i] = 1; // Mark it used
      return pool + (i * sizeof(framis));
    }
  out << "out of memory" << endl;
  return 0;
}

void framis::operator delete(void* m) {
  if(!m) return; // Check for null pointer
  // Assume it was created in the pool
  // Calculate which block number it is:
  unsigned long block = (unsigned long)m
    - (unsigned long)pool;
  block /= sizeof(framis);
  out << "freeing block " << block << endl;
  // Mark it free:
  alloc_map[block] = 0;
}

main() {
  framis* f[framis::psize];
  for(int i = 0; i < framis::psize; i++)
    f[i] = new framis;
  new framis; // Out of memory
  delete f[10];
  f[10] = 0;
  // Use released memory:
  framis* x = new framis;
  delete x;
  for(int j = 0; j < framis::psize; j++)
    delete f[j]; // Delete f[10] OK
}
```

[illegible] psize [illegible] framis[illegible] framis[illegible] psize[illegible] 0[illegible] 0[illegible]

[illegible] new[illegible] new[illegible] 0[illegible] 1[illegible] 0[illegible] new-handler[illegible] new[illegible] delete[illegible] iostreams[illegible]

[illegible] delete[illegible] framis[illegible] framis[illegible] ——[illegible] new[illegible] new[illegible] delete[illegible]

□□□ delete□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□ 0□□□□□□□□□□□□□

□ main( )□□□□□□□□□□ framis□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□ framis□□□□□□□□□□□□ new□ delete□□□□□ □□□□□□□□□□□□□□

12.5.3 □□□□□ new□ delete

□□□□□□□□□□□□ new□ delete□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□ new( )□□□□□□□□ □□□□□□□□□□□ delete( )□□□□□□□□□□□□□□□□□□□□□□□□□ □□□ new[]□ delete[]□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□

```
//: NEWARRY.CPP -- Operator new for arrays
#include <new.h> // Size_t definition
#include <fstream.h>
ofstream trace("newarry.out");

class widget {
  int i[10];
public:
  widget() { trace << "*"; }
  ~widget() { trace << "~"; }
  void* operator new(size_t sz) {
    trace << "widget::new: "
         << sz << " bytes" << endl;
    return ::new char[sz];
  }
  void operator delete(void* p) {
    trace << "widget::delete" << endl;
    ::delete []p;
  }
  void* operator new[](size_t sz) {
    trace << "widget::new[]: "
         << sz << " bytes" << endl;
    return ::new char[sz];
  }
  void operator delete[](void* p) {
    trace << "widget::delete[]" << endl;
    ::delete []p;
  }
};

main() {
  trace << "new widget" << endl;
  widget* w = new widget;
  trace << "\ndelete widget" << endl;
  delete w;
  trace << "\nnew widget[25]" << endl;
  widget* wa = new widget[25];
  trace << "\ndelete []widget" << endl;
  delete []wa;
};
```

[illegible] new[illegible] delete[illegible] new [illegible] delete[illegible] new[illegible] delete[illegible]

[illegible] new[illegible] delete[illegible] new[illegible]

[illegible]

```
new widget
widget::new: 20 bytes
*
delete widget
~widget::delete

new widget[25]
widget::new[]: 504 bytes
*************************
delete []widget
~~~~~~~~~~~~~~~~~~~~~~~~~widget::delete[]
```

[illegible] int[illegible] 2[illegible] 20[illegible] new[illegible] *[illegible] delete[illegible] delete[illegible]

[illegible] widget[illegible] new[illegible] 4[illegible] 4[illegible]

```
delete []widget;
```

[illegible] new[illegible] delete[illegible]

12.5.4 [illegible]

```
foo* f = new foo;
```

[illegible] new[illegible] foo[illegible] foo[illegible]

[illegible] new[illegible] 0[illegible] 0[illegible]

```
//: NOMEMORY.CPP -- Constructor isn't called
// If new returns 0
#include <iostream.h>
#include <new.h> // size_t definition
void my_new_handler() {
  cout << "new handler called" << endl;
}

class nomemory {
public:
  nomemory() {
    cout << "nomemory::nomemory()" << endl;
  }
  void* operator new(size_t sz) {
    cout << "nomemory::operator new" << endl;
    return 0; // "Out of memory"
  }
};

main() {
  set_new_handler(my_new_handler);
  nomemory* nm = new nomemory;
  cout << "nm = " << nm << endl;
}
```

[illegible] new[illegible] new[illegible] 0[illegible]

12.5.5 [illegible]

[illegible] new[illegible]

1) [illegible]

2) [illegible] new[illegible]

[illegible] new[illegible]

[illegible] new[illegible] new[illegible] size_t[illegible]

```
X* xp = new(a) X ;
```

[illegible] a[illegible] new[illegible] new[illegible]

[illegible]

```
//: PLACEMNT.CPP -- Placement with operator new
#include <stddef.h> // Size_t
#include <iostream.h>

class X {
  int i;
public:
  X(int I = 0) { i = I; }
  ~X() {
    cout << "X::~X()" << endl;
  }
  void* operator new(size_t, void* loc) {
    return loc;
  }
};
main() {
  int l[10];
  X* xp = new(l) X(47); // X at location l
  xp->X::~X(); // Explicit destructor call
  // ONLY use with placement!
}
```

[illegible] new[illegible] new[illegible]

[illegible] delete[illegible] “[illegible]”[illegible]

[illegible]

xp->X::~X( ) ; //explicit destructor call

[illegible] new [illegible]

[illegible]

12.6 [illegible]

[illegible]

[illegible] C[illegible] C++[illegible] new[illegible] delete[illegible] C++[illegible] new[illegible] delete[illegible] 17[illegible]

12.7 [illegible]

1. [illegible] cout[illegible] new[illegible] delete[illegible] new[illegible] delete[illegible]

2. [illegible] pstash[illegible] 1[illegible] new[illegible]

3. [illegible] new[illegible] delete[illegible]

4. [illegible] FRAMIS.CPP[illegible] new[illegible] delete[illegible] new[illegible] delete [illegible]

□□

□ 13□ □□□□□

C++□□□

□ C□□□□□□□□□□□□□□□□□□□□□ C++□□

□□

□□□

□□

13.1 □□□□

□□

□□□□□□□□□□□□□□□□□□□

```
//: USEFUL.H -- A class to reuse
#ifndef USEFUL_H_
#define USEFUL_H_

class X {
  int i;
  enum { factor = 11 };
public:
  X() { i = 0; }
  void set(int I) { i = I; }
  int read() const { return i; }
  int permute() { return i = i * factor; }
};
#endif // USEFUL_H_
```

□□□□□□□□□□□□□□□□□□□□□□ X □□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
//: COMPOSE.CPP -- Reuse code with composition
#include "..\12\useful.h"

class Y {
  int i;
public:
  X x; // Embedded object
  Y() { i = 0; }
  void f(int I) { i = I; }
  int g() const { return i; }
};

main() {
  Y y;
  y.f(47);
  y.x.set(37); // Access the embedded object
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□private□□□public□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
//: COMPOSE2.CPP -- Private embedded objects
#include "..\12\useful.h"
class Y {
  int i;
  X x; // Embedded object
public:
  Y() { i = 0; }
  void f(int I) { i = I; x.set(I); }
  int g() const { return i * x.read(); }
  void permute() { x.permute(); }
};

main() {
  Y y;
  y.f(47);
  y.permute();
}
```

□□□permute()□□□□□□□□X□□□□□X□□□□□□□□□Y□□□□□□□□□□

13.2 □□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□“□□□□□□□□□□□”□□□

□Y□□□□□□□□□□□□□□Y□□□X□□□□□□□□□□□□□□□□

```
//: INHERIT.CPP -- Simple inheritance
#include "..\12\useful.h"
#include <iostream.h>
class Y : public X {
  int i; // Different from X's i
public:
  Y() { i = 0; }
  int change() {
    i = permute(); // Different name call
    return i;
  }
  void set(int I) {
    i = I;
    X::set(I); // Same-name function call
  }
};

main() {
  cout << "sizeof(X) = " << sizeof(X) << endl;
  cout << "sizeof(Y) = "
       << sizeof(Y) << endl;
  Y D;
  D.change();
  // X function interface comes through:
  D.read();
  D.permute();
  // Redefined functions hide base versions:
  D.set(12);
}
```

[illegible] Y [illegible] X [illegible] Y [illegible] X [illegible] X [illegible] main() [illegible] sizeof(Y [illegible] sizeof(X) [illegible]

[illegible] public [illegible] private [illegible] public [illegible] private [illegible] public [illegible] public [illegible] public [illegible]

[illegible] change() [illegible] permute() [illegible] public [illegible]

[illegible] set() [illegible] set() [illegible] Y [illegible] read() [illegible] permute() [illegible] main() [illegible] Y [illegible] set() [illegible] change() [illegible]

[illegible] set() [illegible]

[illegible]

13.3 [illegible]

[illegible] C++[illegible] private[illegible]

[illegible] C++[illegible] bar [illegible] foo[illegible] bar [illegible] int[illegible]

```
foo::foo(int i) : bar(i) { //...
```

13.3.1 [illegible]

[illegible]

```
foo2:foo2(int I) : bar(i), memb(i+1) { // ...
```

[illegible] foo2 [illegible] bar [illegible] memb [illegible]

13.3.2 [illegible]

[illegible] C++[illegible]

[illegible]

[illegible]

```
class X {
  int i;
  float f;
  char c;
  char* s;
public:
  X() : i(7), f(1.4), c('x'), s("howdy") {}
  // ...
```

□ □ “ □ □ □ □ □ □ □ ” □

□ □

```
int  i(100);
```

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □

□ □

13.4 □ □ □ □ □ □ □ □

□ □

```
//: COMBINED.CPP -- Inheritance & composition

class A {
  int i;
public:
  A(int I) { i = I; }
  ~A() {}
  void f() const {}
};

class B {
  int i;
public:
  B(int I) { i = I; }
  ~B() {}
  void f() const {}
};

class C : public B {
  A a;
public:
  C(int I) : B(I), a(I) {}
  ~C() {} // Calls ~A() and ~B()
  void f() const {  // Redefinition
    a.f();
    B::f();
  }
};

main() {
  C c(47);
}
```

C □□ B □□□□□□□□□□□□□ A□□

□□C::f()□□□□□□□□□ B::f()□□□□□□□□□□□□□□□□□□□□□□ a.f()□□□C::f()□□□□□□□□□□□□□□ C □□□□□□□ f()□□□□ a.f()□□□□□ B::f()□

- □□□□□□□□□□

□□□

□□

13.4.1 □□□□□□□□□□□□

□□□

```
//: ORDER.CPP -- Constructor/destructor order
#include <fstream.h>
ofstream out("order.out");

#define CLASS(ID) class ID { \
public: \
  ID(int) { out << #ID " constructor\n"; } \
  ~ID() { out << #ID " destructor\n"; } \
};

CLASS(base1);
CLASS(member1);
CLASS(member2);
CLASS(member3);
CLASS(member4);

class derived1 : public base1 {
  member1 m1;
  member2 m2;
public:
  derived1(int) : m2(1), m1(2), base1(3) {
    out << "derived1 constructor\n";
  }
  ~derived1() {
    out << "derived1 destructor\n";
  }
};
```

```
class derived2 : public derived1 {
  member3 m3;
  member4 m4;
public:
  derived2() : m3(1), derived1(2), m4(3) {
    out << "derived2 constructor\n";
  }
  ~derived2() {
    out << "derived2 destructor\n";
  }
};

main() { derived2 d2; }
```

[illegible] ofstream [illegible] 18[illegible]

```
base1 constructor
member1 constructor
member2 constructor
derived1 constructor
member3 constructor
member4 constructor
derived2 constructor
derived2 destructor
member4 destructor
member3 destructor
derived1 destructor
member2 destructor
member1 destructor
base1 destructor
```

[illegible] —— [illegible]

13.4.2 [illegible]

[illegible]

□□□□□□□□□□□□□□□□□□□□□□

```
//: HIDE.CPP -- Name hiding during inheritance

class homer {
public:
  int doh(int) const { return 1; }
  char doh(char) const { return 'd';}
  float doh(float) const { return 1.0; }
};

class bart : public homer {
public:
  class milhouse {};
  void doh(milhouse) const {}
};

main() {
  bart b;
//! b.doh(1); // Error
//! b.doh('x'); // Error
//! b.doh(1.0); // Error
}
```

□□ bart □□□□ doh()□□□□□□□□□□□□□□□ bart □□□□□□□□□,□□□□□□□□□□□□□ milhouse □□□

13.4.3 □□□□□□□□

□□

□□□ operator= □□ 11 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ operator= □□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
//: NINHERIT.CPP -- Non-inherited functions
#include <fstream.h>
ofstream out("ninherit.out");

class root {
public:
  root() { out << "root()\n"; }
  root(root&) { out << "root(root&)\n"; }
  root(int) { out << "root(int)\n"; }
  root& operator=(const root&) {
    out << "root::operator=()\n";
    return *this;
  }
  class other {};
  operator other() const {
    out << "root::operator other()\n";
    return other();
  }
  ~root() { out << "~root()\n"; }
};

class derived : public root {};

void f(root::other) {}

main() {
  derived d1;  // Default constructor
  derived d2 = d1; // Copy-constructor
//! derived d3(1); // Error: no int constructor
  d1 = d2; // Operator= not inherited
  f(d1); // Type-conversion IS inherited
}
```

□□□□□□□□ operator= □□□□□□□□□□□□□□□□□□□□□□□□□□ operator other() □□□□□□□□□□ root □□□□□□□□ other □□□□□ derived □□□ root □□□□□□□□□□□□□□□□□□□□□□□□ f() □□□ other □□□□□□□□□□□□□□□□

□ main() □□□□□□□□□□□□□□□□□□□□ root □□

□ derived □□ operator=( ) □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□——□□□ root □□□□□□□□□□ other □□□□□□ root □□

13.5 □□□□□□□□

□□

China-pub.com [illegible]

[illegible] private [illegible]

[illegible] public[illegible] Car [illegible]

```
//: CAR.CPP -- Public composition

class engine {
public:
  void start() const {}
  void rev() const {}
  void stop() const {}
};

class wheel {
public:
  void inflate(int psi) const {}
};

class window {
public:
  void rollup() const {}
  void rolldown() const {}
};

class door {
public:
  window Window;
  void open() const {}
  void close() const {}
};

class car {
public:
  engine Engine;
  wheel Wheel[4];
  door left, right; // 2-door
};

main() {
  car Car;
  Car.left.Window.rollup();
  Car.Wheel[0].inflate(72);
}
```

[illegible]；[illegible]

[illegible] —— [illegible] is-a[illegible] has-a [illegible]

13.5.1 [illegible]

[illegible] ifstream [illegible] ifstream [illegible] strstream [illegible]

```
//: FNAME1.CPP -- An fstream with a file name
#include <fstream.h>
#include <strstrea.h>
#include "..\allege.h"

class fname1 {
  ifstream File;
  enum { bsize = 100 };
  char buf[bsize];
  ostrstream Name;
  int nameset;
public:
  fname1() : Name(buf, bsize), nameset(0) {}
  fname1(const char* filename)
    : File(filename), Name(buf, bsize) {
      allegefile(File);
      Name << filename << ends;
      nameset = 1;
  }
  const char* name() const { return buf; }
  void name(const char* newname) {
    if(nameset) return; // Don't overwrite
    Name << newname << ends;
    nameset = 1;
  }
  operator ifstream&() { return File; }
};
main() {
  fname1 file("fname1.cpp");
  cout << file.name() << endl;
  // Error: rdbuf() not a member:
//!  cout << file.rdbuf() << endl;
}
```

[illegible] fname1 [illegible] ifstream &[illegible] ifstream [illegible] fname1 [illegible] main [illegible] cout<<file.rdbuf()<<endl;

□□

□□□□□□□ fname1 □□ rdbuf() □□□

filebuf * rdbuf() {return File.rdbuf();}

□□□□□□□□□□ ifsream □□

□□

```
//: FNAME2.CPP -- Subtyping solves the problem
#include <fstream.h>
#include <strstrea.h>
#include "..\allege.h"
class fname2 : public ifstream {
  enum { bsize = 100 };
  char buf[bsize];
  ostrstream Name;
  int nameset;
public:
  fname2() : Name(buf, bsize), nameset(0) {}
  fname2(const char* filename)
    : ifstream(filename), Name(buf, bsize) {
      Name << filename << ends;
      nameset = 1;
  }
  const char* name() const { return buf; }
  void name(const char* newname) {
    if(nameset) return; // Don't overwrite
    Name << newname << ends;
    nameset = 1;
  }
};

main() {
  fname2 file("fname2.cpp");
  allegefile(file);
  cout << "name: " << file.name() << endl;
  const bsize = 100;
  char buf[bsize];
  file.getline(buf, bsize); // This works too!
  file.seekg(-200, ios::end);
  cout << file.rdbuf() << endl;
}
```

[illegible] ofstream [illegible] fname2 [illegible] fname2 [illegible] ofstream[illegible] 14[illegible]

13.5.2 [illegible]

[illegible]

[illegible] stack [illegible] ——[illegible]

[illegible] stack [illegible]

```
//: INHSTAK.CPP -- Specializing the stack class
#include "..\11\stack11.h"
#include "..\11\strings.h"
#include <fstream.h>
#include "..\allege.h"

class Stringlist : public stack {
public:
  void push(String* str) {
    stack::push(str);
  }
  String* peek() const {
    return (String*)stack::peek();
  }
  String* pop() {
    return (String*)stack::pop();
  }
};

main() {
  ifstream file("inhlist.cpp");
  allegefile(file);
  const bufsize = 100;
  char buf[bufsize];
  Stringlist textlines;
  while(file.getline(buf,bufsize))
    textlines.push(String::make(buf));
  String* s;
  while((s = textlines.pop()) != 0) // No cast!
    cout << *s << endl;
}
```

[illegible] STRINGS.H[illegible] 12.2.1[illegible] STACK11.H [illegible] 12.2.3[illegible] 12[illegible]

□ STACK11.OBJ □□□□□□□□□□□

stringlist□□□ stack □□□ push()□□□ string□□□□□□□□ stack□□□ void□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ peek()□ pop()□□□□ string□□□□□□□ void□□□□□□□□□□□□□□□□□

□□□

□□ 12□□□ stack::~stack()□□□□□□□□□□□□□□□□□□□ delete□□□□□□□ void□□□□ delete□□□□□□□□□□□□□□□□□□□□□ void□□□□□□□□□□□□□□□ stringlist::~stringlist()□□□□□□□□□□□□□□□□□□□□ string□□□□ delete□□□□□□□□□□□□□□□

1) stack□□□□□□□□ protected□□□□□ stringlist□□□□□□□□□□□ protected□□□□□□□□□

2) □□ stack□□□□□□□□□□□□□□□□□□□□□

3) □□ string□□□□□□ stringlist□□□□□□□

□□□□□□□□□□□□□□□□ 15□□□□□□□□□□□□□□□□

13.5.3 □□□□

□□□□□□□□□ public □□□□□□□□□ private□□□ 13.5.1□□□ FNAME2.CPP□□□□

□□□□□□□ private □□□□□□□□□□□□□□□□□□□□□□□ private□□□□□□□□□ private□□□ private□□□□□ private□□ private□□□□□□□□□□

- □□□□□□□□□□

□□□□□□□□□□□□ public□□□□□□ private□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ public□□□□□□□□□□□□□□

```
//: PRIVINH.CPP -- Private inheritance

class base1 {
public:
  char f() const { return 'a'; }
  int g() const { return 2; }
  float h() const { return 3.0; }
};

class derived : base1 { // Private inheritance
public:
  base1::f; // Name publicizes member
  base1::h;
};

main() {
  derived d;
  d.f();
  d.h();
//! d.g(); // Error -- private function
}
```

□ private□ □ □ □ □ □ □

□ □ □ □ □ private□ □□ □ 18□ □ □ □□

13.6 □ □

□ □ □ protected□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ private□ protected□ □ □ □ □ □ □ □ □ □ □ □□“ □ □ □ □ □ □ □ □ □ □ □ □ private□ □”

□ □ □ □ □ □ □ private□ □

```
//: PROTECT.CPP -- The protected keyword
#include <fstream.h>

class base {
  int i;
protected:
  int read() const { return i; }
  void set(int I) { i = I; }
public:
  base(int I = 0) : i(I) {}
  int value(int m) const { return m*i; }
};

class derived : public base {
  int j;
public:
  derived(int J = 0) : j(J) {}
  void change(int x) { set(x); }
};

main() {}
```

□ □ □ C□ □ SSHAPE□ □ □ □ □ □ □ □ □ □ □ □ protected□ □ □ □ □ □ □

[illegible]

[illegible] private[illegible] public[illegible] private[illegible] public[illegible] protected[illegible]

[illegible]“[illegible]”[illegible]“ is-a”[illegible]

13.7 [illegible]

[illegible]

[illegible] 16[illegible]

13.8 [illegible]

[illegible]

[illegible]

[illegible]

[illegible]“[illegible]”[illegible]

[illegible]“[illegible]”[illegible]

13.9 [illegible]

[illegible] ofstream [illegible] ofstream [illegible] 13.5.1[illegible] FNAME2.CPP[illegible] ofstream[illegible] fname2[illegible]

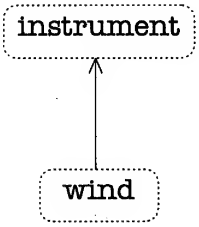
[illegible]“[illegible]”[illegible]

[illegible] —— [illegible]

instrument□ □□□□□□□□□□□□□ wind□ □□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ instrument□ □ play()□ □□□□□□ wind □□□□□□□□□□□□□□□□□ wind□ instrument □□□ □□□□□□□□□□□□□□□□□□□□□□□□

```
//: WIND.CPP -- Inheritance & upcasting
enum note { middleC, Csharp, Cflat }; // Etc.

class instrument {
public:
  void play(note) const {}
};

// Wind objects are instruments
// because they have the same interface:
class wind : public instrument {};

void tune(instrument& i) {
  // ...
  i.play(middleC);
}

main() {
  wind flute;
  tune(flute); // Upcasting
}
```

□□□□□□□□□□□ tune()□ □□□□□ instrument□ □□□□□□ main()□ □ tune()□ □□□□□□□□□□□ wind□□ □□□□□□□□□□□ C++□ □□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ wind□ □□ □□□ instrument□ □□ tune()□ □□□ instrument □□□□□□ wind□ □□□□□□□□□□□□ tune()□ □□□□□□ instrument□ □ instrument□ □□□□□□□□□□□□□□ wind□ □□□ □□□□□□□□ instrument□ □□□□□□□□□□□□□□□□□



13.9.1 □□□ “□□□□”

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□(□□ WIND.CCP□ □□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□□□ □□

- □□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□ 17□ □□□□□□□

13.9.2 □ □ □ □ □

□ stack□ □ □ □ □ □ □ □ □ □ □ □ □ stringlist□ □ □ □ □ string□ □

```
//: INHSTAK2.CPP -- Composition vs inheritance
#include "..\11\stack11.h"
#include "..\11\strings.h"
#include <fstream.h>
#include "..\allege.h"

class Stringlist {
  stack Stack; // Embed instead of inherit
public:
  void push(String* str) {
    Stack.push(str);
  }
  String* peek() const {
    return (String*)Stack.peek();
  }
  String* pop() {
    return (String*)Stack.pop();
  }
};

main() {
  ifstream file("inhlst2.cpp");
  allegefile(file);
  const bufsize = 100;
  char buf[bufsize];
  Stringlist textlines;
  while(file.getline(buf,bufsize))
    textlines.push(String::make(buf));
  String* s;
  while((s = textlines.pop()) != 0) // No cast!
    cout << *s << endl;
}
```

□ □ □ □ □ INHSTACK.CPP(13.5.2□ □)□ □ □ □ □ □ □ □ stack□ □ □ □ □ □ stringlist□ stringlist□ □

□ □ □ □ □ □ private□ □ □ □ □ “ □ □ □ □ ”□ □ □ □ □ stringlist□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ——□ □

13.9.3 □□□□□□□□□□

□ WIND.CPP(13.9)□□□□□□□□□□□□□□□——□□□□□ wind□□□□□□□□□□□□□□□□□□ instrument□□□□

□□□□□□□□□□□□□□□□□□□□□□

```
wind w;
instrument* ip = &w; // upcast
instrument& ir = w; // upcast
```

□□□□□□□□□□□□□□□□□□□□□□□

13.9.4 □□

□□□□□□□□□□□□□□□□□□□□□□□□

```
wind w;
instrument * ip = &w;
```

□□□□□□ ip□□□□ instrument□□□□□□□□□□□□□ ip□□□□□ wind□□□□□□□□□□ play()□□□□□□□□□□

```
ip->play(middleC);
```

□□□□□□□□□□□□□ instrument□□□ play()□□□ instrument:: play()□□□□□□□□□□□□□□□ wind:: play()□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ C++ □□ virtual□□□□□□□□

13.10 □□

□□□

□□

□□

13.11 □□

1. □□ CAR.CPP□□□□□□□□□ vehicle□□□□□□ vehicle□□□□□□□□□□□□□□□□□□□□□□□□□□□ vehicle□□□□□□□□□□□□□□ car□□□□□□□□□□□□□

2. □□□□□□ A□ B□□□□□□□□□□□□□□□□□ A□□□□□□□□□□ C□□□□ C□□□ B□□□□□□□□□□□ C□□□□□□□□□□ C□□□□□□□□□□

3. □□□□□□□□□□ 12□□ PSTASH.H & PSTASH.CPP□□□ pstash□□□□□□□□□□□ String□□□□□ PSTEST.CPP□□□□□□□□□□□□ pstash□□□□□□□□

4. □□ private□ protected□□□

[illegible]14[illegible] [illegible]

[illegible]C++[illegible]

[illegible]“ what” [illegible] “ how” [illegible]“ [illegible] ”[illegible]“ [illegible] ”[illegible]

[illegible] private[illegible]

[illegible]

- C++[illegible]

C[illegible] C++[illegible]

[illegible] C++[illegible]“ [illegible] C”[illegible] C++[illegible] C++[illegible] C[illegible]

[illegible]“ [illegible] ” [illegible] C++[illegible] C[illegible] C++[illegible]

[illegible]——[illegible]

[illegible]

[illegible] C++[illegible] OOP[illegible]

[illegible]

14.1 [illegible]

[illegible]

□□

□□□□□□□□□□□□□□□□□□□□□□□

```
//: WIND2.CPP -- Inheritance & upcasting
#include <iostream.h>
enum note { middleC, Csharp, Cflat }; // Etc.

class instrument {
public:
  void play(note) const {
    cout << "instrument::play" << endl;
  }
};

// Wind objects are instruments
// because they have the same interface:
class wind : public instrument {
public:
  // Redefine interface function:
  void play(note) const {
    cout << "wind::play" << endl;
  }
};

void tune(instrument& i) {
  // ...
  i.play(middleC);
}

main() {
  wind flute;
  tune(flute); // Upcasting
}
```

□□tune()□□□□□□□□□□ instrument□□□□□□□□□ instrument□□□□□□ main()□□□□□□□□□□□□□□ wind□□□□ tune()□□□□□□□□□□ instrument □□□□□□□□□ wind□□□□ wind□□□□□ instrument□□□□□□ wind□ instrument□□□□□□□ wind□□□“ □□ ”□□□□□□□□ instrument□□□□□□□

□□□

14.2 □□

WIND2.CPP□□□□□□□□□□□□□□□□□□□ instrument::play□□□□□□□

□□□□□□□□□□□□□□□□□□□□ wind□ □□□□□ instrument□ □□□□□□□ □ wind::play□ □□□□ instrument□ □□□□□□□□□□ play□ □□□□□

□□□□□□□□ C□□□□ WIND2.CPP□ □□□□□□□□□□□□□□□□□□□□ □□□□□□

□□□□□□

□□□□□□□□□□□□□□□□□ binding□□ □□□□□□□□□□□□□□□□□ □□□ C□ □□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□ instrument□ □□□□□□□□□ □□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□ □□□ □□□ □□□□□□□□□□□□□□

14.3 □□□

□□□□□□□□□□□□□□□□ C++□ □□□□□□□□□□□□□□ virtual□ □□□ □□□□□ virtual□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□ virtual□ □□□□□□□□□□□□□□□□□□

□□□□□□ virtual□ □□□□□□□□□□□□□□□□□□□□□□□□□ virtual□ □ □□ □□□□□□□□□□□□□ virtual□ □□□□□□□□□□□□□ virtual□ □□□□□□ virtual□ □□□□□□□□□□□□

□□□ WIND2.CPP□ □□□□□□□□□□□□□□□□□□ play()□ □□□ virtual□ □□□

```
//: WIND3.CPP -- Late binding with virtual
#include <iostream.h>
enum note { middleC, Csharp, Cflat }; // Etc.

class instrument {
public:
  virtual void play(note) const {
    cout << "instrument::play" << endl;
  }
};

// Wind objects are instruments
// because they have the same interface:
class wind : public instrument {
public:
    // Redefine interface function:
    void play(note) const {
      cout << "wind::play" << endl;
    }
  };

  void tune(instrument& i) {
    // ...
    i.play(middleC);
  }

  main() {
    wind flute;
    tune(flute); // Upcasting
  }
```

□□□□□□□□□ virtual□□□□□□□□ WIND2.CPP□□□□□□□□□□□□□□□□□□□□ wind::play□

□□□

□□□ play( )□□□□□□□ virtual□□□□□ tune()□□□□□□□□□□□□□□□□□□□□□□□□□□ OOP□□□□□□□□□□□□□□□ tune()□□□

□□□ instrument□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ tune()□□□

```
  //: WIND4.CPP -- Extensibility in OOP
  #include <iostream.h>
  enum note { middleC, Csharp, Cflat }; // Etc.

  class instrument {
  public:
    virtual void play(note) const {
      cout << "instrument::play" << endl;
    }
    virtual char* what() const {
      return "instrument";
    }
    // Assume this will modify the object:
    virtual void adjust(int) {}
  };

  class wind : public instrument {
  public:
    void play(note) const {
```

China-pub.com 下载

```
    cout << "wind::play" << endl;
  }
  char* what() const { return "wind"; }
  void adjust(int) {}
};

class percussion : public instrument {
public:
  void play(note) const {
    cout << "percussion::play" << endl;
  }
  char* what() const { return "percussion"; }
  void adjust(int) {}
};
class string : public instrument {
public:
  void play(note) const {
    cout << "string::play" << endl;
  }
  char* what() const { return "string"; }
  void adjust(int) {}
};

class brass : public wind {
public:
  void play(note) const {
    cout << "brass::play" << endl;
  }
  char* what() const { return "brass"; }
};

class woodwind : public wind {
public:
  void play(note) const {
    cout << "woodwind::play" << endl;
  }
  char* what() const { return "woodwind"; }
};

// Identical function from before:
void tune(instrument& i) {
  // ...
  i.play(middleC);
}

// New function:
```

```
void f(instrument& i) { i.adjust(1); }

// Upcasting during array initialization:
instrument* A[] = {
  new wind,
  new percussion,
  new string,
  new brass
};

main() {
  wind flute;
  percussion drum;
  string violin;
  brass flugelhorn;
  woodwind recorder;
  tune(flute);
  tune(drum);
  tune(violin);
  tune(flugelhorn);
  tune(recorder);
  f(flugelhorn);
}
```

[illegible] wind[illegible] virtual[illegible] adjust()[illegible] brass[illegible] woodwind[illegible] ——[illegible]

[illegible] A[][illegible] instrument [illegible] f()[illegible]

[illegible] tune()[illegible] “ [illegible] ”[illegible] virtual[illegible] C++[illegible] C[illegible]

14.4 C++[illegible]

[illegible] C++[illegible]

[illegible] virtual[illegible] brass[illegible] instrument[illegible] play()[illegible]

China-pub.com [illegible]

[illegible]

[illegible] VTABLE[illegible] VTABLE[illegible] vpointer[illegible] VPTR[illegible] VTABLE[illegible] VPTR[illegible] VTABLE[illegible]

[illegible] VTABLE[illegible] VPTR[illegible]

[illegible]

14.4.1 [illegible]

[illegible]

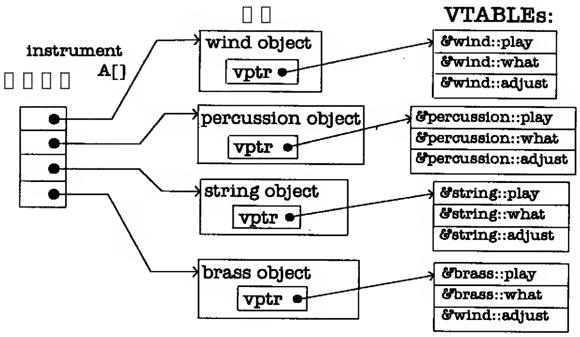
```
//: SIZES.CPP -- Object sizes vs. virtual funcs
#include <iostream.h>

class no_virtual {
  int a;
public:
  void x() const {}
  int i() const { return 1; }
};

class one_virtual {
  int a;
public:
  virtual void x() const {}
  int i() const { return 1; }
};
class two_virtuals {
  int a;
public:
  virtual void x() const {}
  virtual int i() const { return 1; }
};

main() {
  cout << "int: " << sizeof(int) << endl;
  cout << "no_virtual: "
       << sizeof(no_virtual) << endl;
  cout << "void* : " << sizeof(void*) << endl;
  cout << "one_virtual: "
```

```
            << sizeof(one_virtual) << endl;
       cout << "two_virtuals: "
            << sizeof(two_virtuals) << endl;
     }
```

□□□□□□□□□□□□□□□□□□□□□□ int□□□□□□□□□□□□□ one_virtual□□□□□□□ no_virtual□□□□□□□ void□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ VPTR□□□ one_virtual □ two_virtuals□□□□□□□□□□□□ VPTR□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□ C++□□“□”□□□□□□□□□□□□□□□□□□□□□□ virtual□□□□□□□□□□□□□□“□”□□□□□□□□□□□□□□□□□□□□□□ int a□□□□□□□□□□□□□□□□□□

14.4.2 □□□□□□


□□
VTABLEs:
instrument
□□□□ A[]
wind object
vptr
&wind::play
&wind::what
&wind::adjust
percussion object
vptr
&percussion::play
&percussion::what
&percussion::adjust
string object
vptr
&string::play
&string::what
&string::adjust
brass object
vptr
&brass::play
&brass::what
&wind::adjust

□ 14-1

□□□ 14.3□ WIND4.CPP□□□□□□□ A[]□

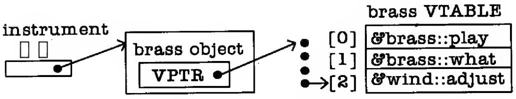
□□ instrument□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ instrument □□□□ wind□ percussion□ string□ brass□□□□□□□□□□□□□□□□□□ instrument□□□□□□□□ instrument □□ instrument□□ virtual□□□ VTABLE□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ virtual□□□□□□□□□□□□□□□□□□□□□□□□□□□□ virtual□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ brass□ VTABLE□□ adjust□□□□□□□□□□□□□□□□□□□□□□□□□□ VPTR□□□ SIZES.CPP□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ VPTR□ VPTR□□□□□□□□□□□□□□ VTABLE□□□□□□□□□□□□□□□□□□□□□□□□□□

□□ VPTR□□□□□□□□□□□□□ VTABLE□□□□“□□”□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□ CALL□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ instrument□□□□ brass□□ adjust()□

instrument□□□□□□□□□□

□□□□□□ instrument□□□□□□□□□□□□□□□□□□□□□□□ instrument □□□□ instrument□□□□□□□□□□ VPTR□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ VPTR□ VPTR□□ VTABLE□□□□□□□□□□ VTABLE□□□□□□□□□□□□□□□□□□□□□□□ play()□□□□□□ what()□□□□□□ adjust()□□


instrument
□□
brass object
VPTR
brass VTABLE
[0]
[1]
[2]
&brass::play
&brass::what
&wind::adjust

□ 14-2

□□□□□□□□□□□□□ adjust()□□□□ VPTR+2□□□□□□□□“□ instrument::adjust□□□□□□□□□”□□□□□□□□□□□□□□□□□□□□□□□□“□ VPTR+2□□□□□□□□”□□□ VPTR□□

14.4.3 □□□□

□□□ f□ instrument&i□□□□

```
i.adjust(1);
```

□□□□□□□□□□□□□□

```
push   1
push   si
mov    bx,word ptr [si]
call   word ptr [bx+4]
add    sp,4
```

C++□□□□□□□□□ C□□□□□□□□□□□□□□□□□□□□□□□□□□□ C□□□□□□□□□□□ 1□□□□□□□□□□□□□□□□□□□□□□□□□□□□ si□ intel x86□□□□□□□□□□□ i□□ this□□□□□□□□□□□□□□□□□□□□□□□ this□□ static□□□□□□□□□□ this□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ VPTR□□□□□□□ VTABLE□□□□□□□□□□□ VPTR□□□□□□□□□□□□ this□□□□□□□ VPTR□□□□□□□

```
mov bx□ word ptr[si]
```

□□ si□□ this□□□□□□□□□□□□ VPTR□□□□□ VPTR□□□□□□ bx□□

□□ bx□□□□□ VPTR□□□□□ VTABLE□□□□□□□□□□□□□□□□□□ VTABLE□□□□□ 0 □□ VPTR□□□ adjust()□□□□□□□□□□□□□□□□□□ VTABLE□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□ VTABLE□□

```
call word ptr [bx+4]
```

[illegible] C [illegible] C++ [illegible]

14.4.4 [illegible] vpointer

[illegible] VPTR [illegible] VPTR [illegible] VTABLE [illegible] VPTR [illegible] WIND [illegible]

[illegible] WIND [illegible] VPTR [illegible] instrument [illegible] instrument [illegible]

[illegible] VPTR [illegible]

14.4.5 [illegible]

[illegible] C++ [illegible]

```
//: EARLY.CPP -- Early binding & virtuals
#include <iostream.h>

class base {
public:
  virtual int f() const { return 1; }
};

class derived : public base {
public:
  int f() const { return 2; }
};
main() {
  derived d;
  base* b1 = &d;
  base& b2 = d;
  base b3;
  // Late binding for both:
  cout << "b1->f() = " << b1->f() << endl;
  cout << "b2.f() = " << b2.f() << endl;
  // Early binding (probably):
  cout << "b3.f() = " << b3.f() << endl;
}
```

[illegible] b1->f() [illegible] b2.f() [illegible] b1 [illegible] b2 [illegible] base [illegible] b3.f() [illegible] base [illegible] base [illegible]

[illegible]

14.5 [illegible]

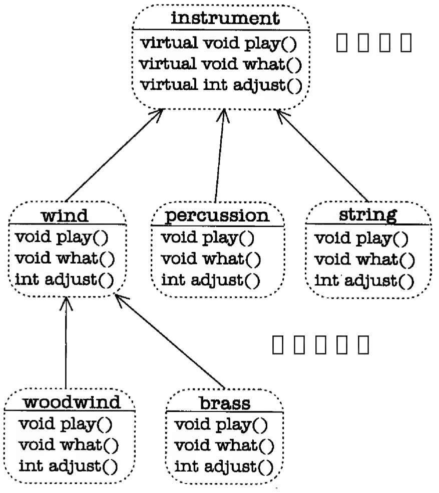
[illegible]“[illegible]‘[illegible]’[illegible]”

[illegible] C++[illegible]“[illegible]”[illegible] CALL[illegible][1] [illegible] C++[illegible] C[illegible] C[illegible]——Unix——[illegible] C++[illegible] C[illegible][2] [illegible] C[illegible] C++[illegible]“[illegible]”[illegible]“[illegible]”[illegible] C[illegible] C++[illegible] VTABLE[illegible] Stroustrup[illegible]“[illegible]”[illegible]

[illegible] virtual[illegible]

[illegible] C++[illegible] C[illegible] 10%[illegible] C++[illegible] C[illegible]

14.6 [illegible]


instrument
virtual void play()
virtual void what()
virtual int adjust()
wind
void play()
void what()
int adjust()
percussion
void play()
void what()
int adjust()
string
void play()
void what()
int adjust()
woodwind
void play()
void what()
int adjust()
brass
void play()
void what()
int adjust()

[illegible] 14-3

[illegible] instrument[illegible] instrument [illegible]“[illegible]”[illegible] instrument[illegible]

[illegible]——[illegible]“[illegible]”[illegible]

[illegible] instrument[illegible]

---

[1] [illegible] Smalltalk [illegible]

[2] [illegible] C++[illegible]

□□□□□□□□□□□□□

□□□□□□□□□□□□□□ virtual□ □□□ virtual□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□ instrument□□ □□□□□□□□□□□□□□□□□□□□□□□ instrument□ □□□□□□□□□□□□□□□□□□□□□□ instrument□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□ instrument□ □□

C++□ □□□□□□□□□□□□□□□□□□□□□□□□□□

```
virtual void x() = 0;
```

□□□□□□□□□□□□ VTABLE□ □□□ VTABLE□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□ VTABLE□ □□□

□□□□□□ WIND4.CPP 14.3□ □□□□□□□□□□

```
//: WIND5.CPP -- Pure abstract base classes
#include <iostream.h>
enum note { middleC, Csharp, Cflat }; // Etc.

class instrument {
public:
  // Pure virtual functions:
  virtual void play(note) const = 0;
  virtual char* what() const = 0;
  // Assume this will modify the object:
  virtual void adjust(int) = 0;
};
// Rest of the file is the same ...

class wind : public instrument {
public:
  void play(note) const {
    cout << "wind::play" << endl;
  }
  char* what() const { return "wind"; }
  void adjust(int) {}
};

class percussion : public instrument {
```

```
public:
  void play(note) const {
    cout << "percussion::play" << endl;
  }
  char* what() const { return "percussion"; }
  void adjust(int) {}
};

class string : public instrument {
public:
  void play(note) const {
    cout << "string::play" << endl;
  }
  char* what() const { return "string"; }
  void adjust(int) {}
};

class brass : public wind {
public:
  void play(note) const {
    cout << "brass::play" << endl;
  }
  char* what() const { return "brass"; }
};

class woodwind : public wind {
public:
  void play(note) const {
    cout << "woodwind::play" << endl;
  }
  char* what() const { return "woodwind"; }
};

// Identical function from before:
void tune(instrument& i) {
  // ...
  i.play(middleC);
}

// New function:
void f(instrument& i) { i.adjust(1); }

main() {
  wind flute;
  percussion drum;
  string violin;
```

China-pub.com [illegible]

```
  brass flugelhorn;
  woodwind recorder;
  tune(flute);
  tune(drum);
  tune(violin);
  tune(flugelhorn);
  tune(recorder);
  f(flugelhorn);
}
```

[illegible]

[illegible]

[illegible] VTABLE[illegible]

[illegible]

[illegible]

```
//: PVDEF.CPP -- Pure virtual base definition
#include <iostream.h>

class base {
public:
  virtual void v() const = 0;
  // In situ:
  virtual void f() const = 0 {
    cout << "base::f()\n";
  }
};

void base::v() const { cout << "base::v()\n";}

class d : public base {
public:
  // Use the common base code:
  void v() const { base::v(); }
  void f() const { base::f(); }
};

main() {
  d D;
  D.v();
  D.f();
}
```

□ base VTABLE□ □

□ □□ □□

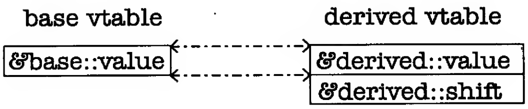
14.7 □ □ □ VTABLE

□ VTABLE□ VTABLE□ □□

□ □

```
//: ADDV.CPP -- Adding virtuals in derivation
#include <iostream.h>

class base {
  int i;
public:
  base(int I) : i(I) {}
  virtual int value() const { return i; }
};
class derived : public base {
public:
  derived(int I) : base(I) {}
  int value() const {
    return base::value() * 2;
  }
  // New virtual function in the derived class:
  virtual int shift(int x) const {
    return base::value() << x;
  }
};

main() {
  base* B[] = { new base(7), new derived(7) };
  cout << "B[0]->value() = "
       << B[0]->value() << endl;
  cout << "B[1]->value() = "
       << B[1]->value() << endl;
//! cout << "B[1]->shift(3) = "
//!      << B[1]->shift(3) << endl; // Illegal
}
```

[illegible] base[illegible] value()[illegible] derived[illegible] shift()[illegible] value[illegible] base[illegible] derived[illegible] VTABLE[illegible]


base vtable
derived vtable
&base::value
&derived::value
&derived::shift

[illegible] 14-4

[illegible] derived VTABLE[illegible] value[illegible] base VTABLE[illegible] derived[illegible] shift[illegible] VTABLE[illegible] derived[illegible] VTABLE[illegible] VTABLE[illegible]

[illegible] value[illegible] derived[illegible] shift[illegible] VTABLE[illegible] VTABLE[illegible]

[illegible]

```
((derived*)B[1])->shift(3)
```

[illegible] B[1][illegible] derived[illegible] RTTI[illegible]

[illegible] “ [illegible] ” [illegible] “ [illegible] ” [illegible]

[illegible] 18[illegible] C++[illegible]

- [illegible]

[illegible] [1][illegible]

[illegible] “ [illegible] ”[illegible]

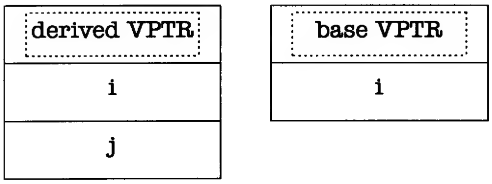
```
//: SLICE.CPP -- Object slicing
#include <iostream.h>
```

---

[1] [illegible]

```
class base {
  int i;
public:
  base(int I = 0) : i(I) {}
  virtual int sum() const { return i; }
};

class derived : public base {
  int j;
public:
  derived(int I = 0, int J = 0)
    : base(I), j(J) {}
  int sum() const { return base::sum() + j; }
};

void call(base b) {
  cout << "sum = " << b.sum() << endl;
}

main() {
  base b(10);
  derived d(10, 47);
  call(b);
  call(d);
}
```

□□call()□□□□□□□□□□□□base□□□□□□□□□□□base□□□□□□□sum()□□□□□□□□□□□□□□□□10□□□□□□□□□57□□□□□□□□□□□□10□



□□□□ □□□□

□ 14-5

□□□□□□□□□□□□□□□□□□□□call()□□□□□□□base□□□□□□□□□□□□□□□□□□□□□□□□□base□□□□□□call()□□□□□□□□□□□□□base□□□□□□□□□□□□□□□□□□□□□□□□□□base□□□□□□□□□□call□□□□□□□□□□□□□□□□□□□□base□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□base□□□□□□□□□□□derived□□□□derived□□□□□□□□□□□□□□□

□□base□□□□□□□□□□□□□□□□□□□□VPTR□□base VTABLE□□□□□□□□□□□□□base□□□base□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

[illegible]

[illegible]

14.8 [illegible]

[illegible] VPTR[illegible] VTABLE[illegible] VPTR[illegible] VPTR[illegible] VPTR[illegible]

[illegible] inline[illegible] C++[illegible] inline[illegible] “[illegible]”[illegible]

[illegible] VPTR[illegible] this[illegible] operator new[illegible]

[illegible]

14.8.1 [illegible]

[illegible]

[illegible] class x[illegible]

[illegible] public [illegible] protected[illegible] “[illegible]” [illegible]

14.8.2 [illegible]

[illegible]

[illegible]

[illegible] —— [illegible] “[illegible]” [illegible] “[illegible]” [illegible]

[illegible] VPTR [illegible] “[illegible]” [illegible] -- [illegible] VPTR [illegible] VTABLE [illegible] VPTR [illegible] VTABLE [illegible] VPTR [illegible] VTABLE [illegible] VPTR [illegible]

[illegible] VPTR [illegible] VTABLE [illegible] VTABLE [illegible] VTABLE [illegible] VTABLE [illegible]

14.9 [illegible]

[illegible] B [illegible]

[illegible] —— [illegible]

[illegible]

[illegible]

□□□□□□□□□□□□□□□□ delete□ □□□□□ new□ □□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□ delete□ □□□□□□□□□□
□□□
□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□“□□”□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ VPTR□□□□
□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□

```
//: PVDEST.CPP -- Pure virtual destructors
// require a function body.
#include <iostream.h>

class base {
public:
  virtual ~base() = 0 {
    cout << "~base()" << endl;
  }
};

class derived : public base {
public:
  ~derived() {
    cout << "~derived()" << endl;
  }
};

main() {
  base* bp = new derived; // Upcast
  delete bp; // Virtual destructor call
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□
□□

□□
□□□□□□□□□□□□□□□□□□□□“□□□”□□□□□□□□□□□□□□□□□□□□
□□□□□□□“□□”□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

[illegible]“[illegible]”[illegible]VPTR[illegible]

14.10 [illegible]

[illegible] C++[illegible]

[illegible] const[illegible] switch[illegible] C++[illegible]

[illegible]

[illegible] C++[illegible] 15[illegible] 17[illegible]

14.11 [illegible]

1. [illegible]“ shape”[illegible] shape[illegible] circle[illegible] square[illegible] triangle[illegible] draw()[illegible] shape[illegible] draw()[illegible]

2. [illegible] 1[illegible] draw()[illegible] shape[illegible] draw()[illegible]

3. [illegible]

4. [illegible] EARLY.CPP[illegible]

5. [illegible] class X[illegible] X [illegible] class Y[illegible] X[illegible] X[illegible]

6. [illegible] 5[illegible] VPTR[illegible]

7. [illegible]

8. [illegible]

□15□ □□□□□□□

□□□□□□□□□□□□□□□□□□□□(building block),□□□□□□□□□□□□
□□□

□□
□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Smalltalk□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C++□□□□
□□□□□□□□□□□□□□□□□□□

□□□□□□□□C++□□□□□□□□□□□□□□Smalltalk□□□□□□□□□□□
□□□□□□□Smalltalk□□□□□□C++□□□□□□□□□□□□□□□□□□□□□
□□□□□

□□
□□□□□□□C++□□□□□□□□□□□□□□□□□□□

□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□
□□□□□□□

15.1 □□□□□□□

□□□□□C□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C++
□□□□□□□□□□□□□□□□□□□□□□stack□□□□□□□□□□□□□□□□□
□□□□□□

```
//: ISTACK.CPP -- Simple integer stack
#include <assert.h>
#include <iostream.h>

class istack {
  enum { ssize = 100 };
  int stack[ssize];
  int top;
public:
  istack() : top(0) { stack[top] = 0; }
  void push(int i) {
    if(top < ssize) stack[top++] = i;
  }
  int pop() {
    return stack[top > 0 ? --top : top];
  }
  friend class istackIter;
};

// An iterator is a "super-pointer":
class istackIter {
  istack& S;
  int index;
public:
  istackIter(istack& is)
    : S(is), index(0) {}
  int operator++() { // Prefix form
    if (index < S.top - 1) index++;
    return S.stack[index];
  }
  int operator++(int) { // Postfix form
    int returnval = S.stack[index];
    if (index < S.top - 1) index++;
    return returnval;
  }
};
// For interest, generate Fibonacci numbers:
int fibonacci(int N) {
  const sz = 100;
  assert(N < sz);
  static F[sz]; // Initialized to zero
  F[0] = F[1] = 1;
  // Scan for unfilled array elements:
  int i;
  for(i = 0; i < sz; i++)
    if(F[i] == 0) break;
  while(i <= N) {
    F[i] = F[i-1] + F[i-2];
    i++;
  }
  return F[N];
}

main() {
  istack is;
  for(int i = 0; i < 20; i++)
    is.push(fibonacci(i));
  // Traverse with an iterator:
  istackIter it(is);
  for(int j = 0; j < 20; j++)
    cout << it++ << endl;
    for(int k = 0; k < 20; k++)
      cout << is.pop() << endl;
  }
```

□ istack□□□

□□□□ istackIter□□□□□□□□□□□□□□□□□□□ istack□□□□□□□□□□□□ istackIter□ istack□□□□□□□□ istack□□□□□□□

□□□□□□□□ istackIter□□□□□□ istack□□□□□□□□□□□□□□□□□□ istackIter□□□□□□□□□□□□ ++□□

□□□□□□□□□□□ fibonacci□□□□□□“□□□□□”□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□ main()□□□

□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ new□□□□□□□ delete□□ CAD□□□

□□□□□□□□□“□□□□□”□ C□□□□□□□ new□ delete□□□□□□□□□□□□□□□□□□□□□□□□□□□□ huge□□□□□□□□□□□□□□□□□□□□□□□□□□ malloc()□ free()□ ,□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□ huge□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ C++□□□□□□□□□□□□□□□□□□ new□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ new□□

15.2 □□□□

□□□□□□□□□□ istack□□

□□□

□□□□□□□□□□□□□□□□ C□□□□□□ C++□ Smalltalk□□□ C++□□□□
□□

15.2.1 C□□

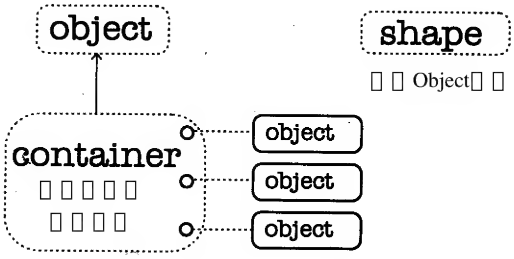
□□□□□□□□□ C□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ stack□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

15.2.2 Smalltalk □□

Smalltalk□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ object□□□□□□ Smalltalk□□ Smalltalk□□□□□□□□ Smalltalk□□□□□□□□□□

□□□□□□ Smalltalk□□□ Smalltalk□□□□□□□□□□ object——□□ Smalltalk□□□□□□□□□□□□

□□□□□□□□□□□□□□ Smalltalk□□□□□□□□□□□ object□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ object□□□□□□□□□□□□□“□□□□□□□”□□□□□□□□□□□□□□□□□□□□□ OOP□□□□□□□□“□□□”□□□□□□□□□□□□□□□ object□□□□□□□□□□□□□□□□□□□□ object□□□□□□


object
shape
□□ Object□□
container
□□□□□
□□□□
object
object
object

□ 15-1

□□ Smalltalk□□□□□□□ C++□□□□□□□□□ C++□□□□□□□□□□□□□ C++□□ Smalltalk□□□□□□□□□□□□□□□□□□□□□□□□□□ C++□□□ [1]□□

□□□□□□□ Smalltalk□□□□□□□□□□□□□□□□□□□□□□□□□□□
C++□□□□□□□□□□□□□□□□□□ object□□□ C□□□□□□□□□□□□□□□□□□□□□□□□□□ object□□□□□□□□□□□□□□□□□□□□□

□□ C++□□□□□□□□□□□□□□ Smalltalk□“□□ object□□□□□”□□□□□□□□

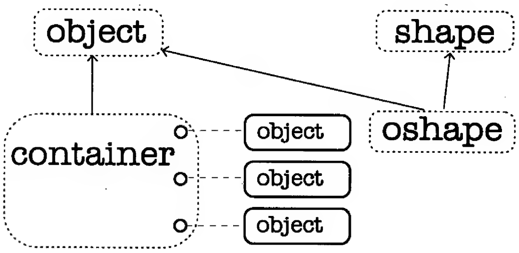
□□□

---

[1] OOPS□□ Keith Gorlen□ NIH□□□□□□□□□□□□□□□□□□□

oshape□□shape□□□□□□□□□□□object□□□□□□□□□□□□□□□□

□□□□□C++□□□□□□□□□□□□□□□□□□□□□□□C++□□□“ C++□□Smalltalk”□□□□□“□□□□□object□□□□□□□□□□□□□”□□□□[1]□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□object□□□


object
shape
container
object
object
object
oshape

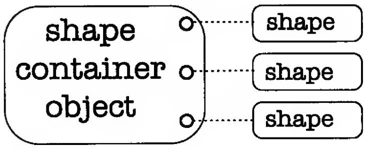
□ 15-2

15.2.3 □□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Stroustrup□□□□□[2]□□□

□□□□□□□□□□□□□Smalltalk□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□Stroustrup□□□□□□□C++□□□[3]□□□object□□


shape
container
object
shape
shape
shape

□ 15-3

□□□

□□C++□□

15.3 □□□□□

“□□□template□”□□□

---

[1] □□□□□□□□□□□□□□□□□□□□□□□□□AT&T□□

[2] The C++Programming Language□□Bjarne Stroustrup□□□□□□Addision-Wesley, 1986□□

[3] □□□□□□□□□□ADA□

□□□□□□□□□□□□□□□:

```
//: STEMP.CPP -- Simple template example
#include <iostream.h>
#include <assert.h>

template<class T>
class array {
  enum { size = 100 };
  T A[size];
public:
  T& operator[](int index) {
    assert(index >= 0 && index < size);
    return A[index];
  }
};
main() {
  array<int> ia;
  array<float> fa;
  for(int i = 0; i < 20; i++) {
    ia[i] = i * i;
    fa[i] = float(i) * 1.414;
  }
  for(int j = 0; j < 20; j++)
    cout << j << ": " << ia[j]
         << ", " << fa[j] << endl;
}
```

□□□□□

template<class T>

□□□□□□□□□□□□□□□□□ T□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□

□ array□□□□□□□□□□□□□□□□□□□□□□ operator[]□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□ C□□□ assert()□□□□□□ □□□□ assert()□□□ allege()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□ 17□□□□□□□□

□ main()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

array<int> ia;

array<float> fa;

□□ □ array_int□ array_float□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□ia□ fa□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

15.3.1 □□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

[illegible]

```
//: STEMP2.CPP -- Non-inline template example
#include <assert.h>

template<class T>
class array {
  enum { size = 100 };
  T A[size];
public:
  T& operator[](int index);
};

template<class T>
T& array<T>::operator[](int index) {
  assert(index >= 0 && index < size);
  return A[index];
}

main() {
  array<float> fa;
  fa[0] = 1.414;
}
```

[illegible] array<T>[illegible]

- [illegible]

[illegible]“[illegible]”[illegible]template<…>[illegible]

[illegible] Windows DLL [illegible] CPP[illegible]

15.3.2 [illegible] (the stack as a template)

[illegible] ISTACK.CPP[illegible] ([illegible] 15.1[illegible])[illegible]

```
//: STACKT.H -- Simple stack template
#ifndef STACKT_H_
#define STACKT_H_
template<class T> class stackIter; // declare

template<class T>
class stackt {
  enum { ssize = 100 };
  T stack[ssize];
  int top;
public:
  stackt() : top(0) { stack[top] = 0; }
  void push(const T& i) {
    if(top < ssize) stack[top++] = i;
  }
  T pop() {
    return stack[top > 0 ? --top : top];
  }
  friend class stacktIter<T>;
};

template<class T>
class stacktIter {
  stackt<T>& S;
  int index;
public:
  stacktIter(stackt<T>& is)
    : S(is), index(0) {}
  T& operator++() { // Prefix form
    if (index < S.top - 1) index++;
    return S.stack[index];
  }
  T& operator++(int) { // Postfix form
    int returnIndex = index;
    if (index < S.top - 1) index++;
    return S.stack[returnIndex];
  }
};
#endif // STACKT_H_
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□ stackt<T>& S□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□ push()□□□□ stackt□□□ T□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□

```
//: STACKT.CPP -- Test simple stack template
#include <assert.h>
#include <iostream.h>
#include "..\14\stackt.h"
// For interest, generate Fibonacci numbers:
int fibonacci(int N) {
  const sz = 100;
  assert(N < sz);
  static F[sz]; // Initialized to zero
  F[0] = F[1] = 1;
  // Scan for unfilled array elements:
  int i;
  for(i = 0; i < sz; i++)
    if(F[i] == 0) break;
  while(i <= N) {
    F[i] = F[i-1] + F[i-2];
    i++;
  }
  return F[N];
}

main() {
  stackt<int> is;
  for(int i = 0; i < 20; i++)
    is.push(fibonacci(i));
  // Traverse with an iterator:
  stacktIter<int> it(is);
  for(int j = 0; j < 20; j++)
    cout << it++ << endl;
  for(int k = 0; k < 20; k++)
    cout << is.pop() << endl;
}
```

 isit

15.3.3

```
//: MBLOCK.CPP -- Built-in types in templates
#include <assert.h>
#include <iostream.h>

template<class T, int size = 100>
class mblock {
  T array[size];
public:
  T& operator[](int index) {
    assert(index >= 0 && index < size);
    return array[index];
  }
};
```

```
class number {
  float f;
public:
  number(float F = 0.0f) : f(F) {}
  number& operator=(const number& n) {
    f = n.f;
    return *this;
  }
  operator float() const { return f; }
  friend ostream&
    operator<<(ostream& os, const number& x) {
      return os << x.f;
  }
};

template<class T, int sz = 20>
class holder {
  mblock<T, sz>* np;
public:
  holder() : np(0) {}
  number& operator[](int i) {
    assert(i >= 0 && i < sz);
    if(!np) np = new mblock<T, sz>;
    return np->operator[](i);
  }
};

main() {
  holder<number, 20> H;
  for(int i = 0; i < 20; i++)
    H[i] = i;
  for(int j = 0; j < 20; j++)
    cout << H[j] << endl;
}
```

□mblock□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□17□□□□□assert()□□□□□□□

□holder□mblock□□□□□□□□holder□□□□□□mblock□□□□□□□□□mblock□□□□□□□□□□□□□□□holder□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□

15.4 stash & stack□□

□□□□□□□□□□□□stash□stack□□

[illegible]

15.4.1 [illegible]

[illegible] C++[illegible]

[illegible]

[illegible]

15.4.2 stash[illegible]

“ stash” [illegible] ,[illegible] ([illegible] 12[illegible])[illegible]

```
//: TSTASH.H -- PSTASH using templates
#ifndef TSTASH_H_
#define TSTASH_H_
#include <stdlib.h>
#include "..\allege.h"
// More convenient than nesting in tstash:
enum owns { no = 0, yes = 1, Default };
// Declaration required:
template<class Type, int sz> class tstashIter;

template<class Type, int chunksize = 20>
class tstash {
  int quantity;
  int next;
  owns own; // Flag
  void inflate(int increase = chunksize);
protected:
  Type** storage;
public:
  tstash(owns owns = yes);
  ~tstash();
  int Owns() const { return own; }
  void Owns(owns newOwns) { own = newOwns; }
  int add(Type* element);
  int remove(int index, owns d = Default);
  Type* operator[](int index);
  int count() const { return next; }
  friend class tstashIter<Type, chunksize>;
};

template<class Type, int sz = 20>
class tstashIter {
  tstash<Type, sz>& ts;
  int index;
public:
  tstashIter(tstash<Type, sz>& TS)
    : ts(TS), index(0) {}
  tstashIter(const tstashIter& rv)
    : ts(rv.ts), index(rv.index) {}
  // Jump interator forward or backward:
  void forward(int amount) {
    index += amount;
    if(index >= ts.next) index = ts.next -1;
  }
  void backward(int amount) {
    index -= amount;
    if(index < 0) index = 0;
  }
  // Return value of ++ and -- to be
  // used inside conditionals:
  int operator++() {
    if(++index >= ts.next) return 0;
    return 1;
  }
  int operator++(int) { return operator++(); }
  int operator--() {
    if(--index < 0) return 0;
    return 1;
  }
int operator--(int) { return operator--(); }
operator int() {
  return index >= 0 && index < ts.next;
}
Type* operator->() {
  Type* t = ts.storage[index];
  if(t) return t;
  allege(0,"tstashIter::operator->return 0");
```

```
    return 0; // To allow inlining
  }
  // Remove the current element:
  int remove(owns d = Default){
    return ts.remove(index, d);
  }
};

template<class Type, int sz>
tstash<Type, sz>::tstash(owns Owns) : own(Owns) {
  quantity = 0;
  storage = 0;
  next = 0;
}

// Destruction of contained objects:
template<class Type, int sz>
tstash<Type, sz>::~tstash() {
  if(!storage) return;
  if(own == yes)
    for(int i = 0; i < count(); i++)
      delete storage[i];
  free(storage);
}

template<class Type, int sz>
int tstash<Type, sz>::add(Type* element) {
  if(next >= quantity)
    inflate();
  storage[next++] = element;
  return(next - 1); // Index number
}

template<class Type, int sz>
int tstash<Type, sz>::remove(int index,owns d){
  if(index >= next || index < 0)
    return 0;
  switch(d) {
    case Default:
      if(own != yes) break;
    case yes:
      delete storage[index];
    case no:
      storage[index] = 0; // Position is empty
  }
  return 1;
```

```
}

template<class Type, int sz> inline
Type* tstash<Type, sz>::operator[](int index) {
  // No check in shipping application:
  assert(index >= 0 && index < next);
  return storage[index];
}

template<class Type, int sz>
void tstash<Type, sz>::inflate(int increase) {
  void* v =
    realloc(storage,
(quantity+increase)*sizeof(Type*));
  allegemem(v);  // Was it successful?
  storage = (Type**)v;
  quantity += increase;
}
#endif // TSTASH_H_
```

□□□□ enum owns□□□

storage□□□ tstash□□□□□□□□□□□□□ SORTED.CPP □□(□ 15.7)□□□□□□□□□□□□□□□

own□□ owns()□□□□□□□□□□□□□

□□□

□□□□□ tstashIter□□ forward() □ backward()□□□ remove()□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□ tstash□□□□□□□□ Int□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 12□□ String□□□□□

```
//: TSTEST.CPP -- Test TSTASH
#include <fstream.h>
#include "..\allege.h"
#include "..\14\tstash.h"
#include "..\11\strings.h"
const bufsize = 80;
ofstream out("tstest.out");
```

```
  class Int {
    int i;
  public:
    Int(int I = 0) : i(I) {
      out << ">" << i << endl;
    }
    ~Int() { out << "~" << i << endl; }
    operator int() const { return i; }
    friend ostream&
      operator<<(ostream& os, const Int& x) {
        return os << x.i;
    }
  };

  main() {
    tstash<Int> intStash; // Instantiate for int
    for(int i = 0; i < 30; i++)
      intStash.add(new Int(i));
    tstashIter<Int> Intit(intStash);
    Intit.forward(5);
    for(int j = 0; j < 20; j++, Intit++)
      Intit.remove(); // Default removal
    for(int k = 0; k < intStash.count(); k++)
      if(intStash[k]) // Remove() causes "holes"
        out << *intStash[k] << endl;

    ifstream file("tstest.cpp");
    allegefile(file);
    char buf[bufsize];
    // Instantiate for String:
  tstash<String> stringStash;
  while(file.getline(buf, bufsize))
    stringStash.add(makeString(buf));
  for(int u = 0; u < stringStash.count(); u++)
    if(stringStash[u])
      out << *stringStash[u] << endl;
  tstashIter<String> it(stringStash);
  int j = 25;
  it.forward(j);
  while(it) {
    out << j++ << ": " << it->str() << endl;
    it++;
  }
}
```

[illegible]

[illegible]

15.4.3 stack[illegible]

[illegible] 13[illegible] stack[illegible]

```
//: TSTACK.H -- Stack using templates
#ifndef TSTACK_H_
#define TSTACK_H_

// Declaration required:
template<class T> class tstackIterator;

template<class T>
class tstack {
  struct link {
    T* data;
    link* next;
    link(T* Data, link* Next) {
      data = Data;
      next = Next;
    }
  } * head;
  int owns;
public:
  tstack(int Owns = 1) : head(0), owns(Owns) {}
  ~tstack();
  void push(T* Data) {
    head = new link(Data,head);
  }
  T* peek() const { return head->data; }
  T* pop();
  int Owns() const { return owns; }
  void Owns(int newownership) {
    owns = newownership;
  }
  friend class tstackIterator<T>;
};

template<class T>
T* tstack<T>::pop() {
  if(head == 0) return 0;
  T* result = head->data;
  link* oldHead = head;
  head = head->next;
  delete oldHead;
  return result;
}

template<class T>
tstack<T>::~tstack() {
  link* cursor = head;
  while(head) {
    cursor = cursor->next;
    // Conditional cleanup of data:
    if(owns) delete head->data;
    delete head;
    head = cursor;
  }
}

template<class T>
class tstackIterator {
  tstack<T>::link* p;
public:
  tstackIterator(const tstack<T>& tl)
    : p(tl.head) {}
  tstackIterator(const tstackIterator& tl)
    : p(tl.p) {}
  // operator++ returns boolean indicating end:
  int operator++() {
    if(p->next)
      p = p->next;
    else p = 0; // Indicates end of list
    return int(p);
  }
  int operator++(int) { return operator++(); }
  // Smart pointer:
  T* operator->() const {
    if(!p) return 0;
    return p->data;
  }
  T* current() const {
    if(!p) return 0;
    return p->data;
  }
  // int conversion for conditional test:
  operator int() const { return p ? 1 : 0; }
};
#endif // TSTACK_H_
```

[illegible] tstash[illegible] Owns()[illegible]

[illegible] tstackIterator[illegible]

[illegible] current()[illegible] 11[illegible] operator int()[illegible]

[illegible] CPP[illegible]

```
//: TSTKTST.CPP -- Use template list & iterator
#include "..\14\tstack.h"
#include "..\11\strings.h"
#include <fstream.h>
#include "..\allege.h"

main() {
  ifstream file("tstktst.cpp");
  allegefile(file);
  const bufsize = 100;
  char buf[bufsize];
  tstack<String> textlines;
  // Read file and store lines in the list:
  while(file.getline(buf,bufsize))
    textlines.push(String::make(buf));
  int i = 0;
  // Use iterator to print lines from the list:
  tstackIterator<String> it(textlines);
  tstackIterator<String>* it2 = 0;
  while(it) {
    cout << *it.current() << endl;
    it++;
    if(++i == 10) // Remember 10th line
      it2 = new tstackIterator<String>(it);
  }
  cout << *(it2->current()) << endl;
  delete it2;
}
```

tstack[illegible] String[illegible]

[illegible]

[illegible] stack[illegible] tstack[illegible]

15.5 [illegible]

[illegible]

15.5.1 [illegible]

[illegible] SString[illegible]

```
//: SSTRING.H -- Stack-based string
#ifndef SSTRING_H_
#define SSTRING_H_
#include <string.h>
#include <iostream.h>

template<int bsz = 0>
class SString {
  char buf[bsz + 1];
  char* s;
public:
  SString(const char* S = "") : s(buf) {
    if(!bsz) { // Make on heap
      s = new char[strlen(S) + 1];
      strcpy(s, S);
    } else { // Make on stack
      buf[bsz] = 0; // Ensure 0 termination
      strncpy(s, S, bsz);
    }
  }
  SString(const SString& rv) : s(buf) {
    if(!bsz) { // Make on heap
      s = new char[strlen(rv.s) + 1];
    strcpy(s, rv.s);
  } else { // Make on stack
    buf[bsz] = 0;
    strncpy(s, rv.s, bsz);
  }
}
SString& operator=(const SString& rv) {
  // Check for self-assignment:
  if(&rv == this) return *this;
```

```
    if(!bsz) { // Manage heap:
      delete s;
      s = new char[strlen(rv.s) + 1];
    }
    // Constructor guarantees length < bsz:
    strcpy(s, rv.s);
    return *this;
  }
  ~SString() {
    if(!bsz) delete []s;
  }
  int operator==(const SString& rv) const {
    return !stricmp(s, rv.s);
  }
  int operator!=(const SString& rv) const {
    return stricmp(s, rv.s);
  }
  int operator>(const SString& rv) const {
    return stricmp(s, rv.s) > 0;
  }
  int operator<(const SString& rv) const {
    return stricmp(s, rv.s) < 0;
  }
  char* str() const { return s; }
  friend ostream&
    operator<<(ostream& os,
               const SString<bsz>& S) {
      return os << S.s;
    }
};

typedef SString<> Hstring; // Heap string
#endif // SSTRING_H_
```

□□□□ typedef Hstring□□□□□□□□□□□□□□□□□□□□□ typedef□□□□□□□□□□□□□□□□□□□□□□□□□□□ =□□ new□ delete□□□□□□□□□□□□□□□□ =□□□□□□□□□□

□□□□□□□□□□□□□□□ stricmp()□□□□□□□□□□ C□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

15.5.2 □□

□ integer□□□□□□□□□□□□□□□□□□□□□□□ int□□□□□□□□□□

□□□□□□□

```
//: INTEGER.H -- Int wrapped in a class
#ifndef INTEGER_H_
#define INTEGER_H_
#include <iostream.h>

class integer {
  int i;
public:
  // Guaranteed zeroing:
  integer(int ii = 0) : i(ii) {}
  operator int() const { return i; }
  const integer& operator++() {
    i++;
    return *this;
  }
  const integer operator++(int) {
    integer returnval(i);
    i++;
    return returnval;
  }
  integer& operator+=(const integer& x) {
    i += x.i;
    return *this;
  }
  friend ostream&
  operator<<(ostream& os, const integer& x) {
      return os << x.i;
  }
};
#endif // INTEGER_H_
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 11□□□□□□□□□□□□□□□□□□□□

15.6 □□

□□ tstash□□ operator[].□

15.6.1 “□□”□□

□□□ operator[]□□□□□□□□□□□□□□□□□□□□□□□□ =□□□

□□□□□□□□□

```
//: VECTOR.H -- "Infinite" vector
#ifndef VECTOR_H_
#define VECTOR_H_
#include <stdlib.h>
#include "..\allege.h"

template<class T>
class vector {
  T** pos;
  int pos_sz;
  T** neg;
  int neg_sz;
  int owns;
  enum {
    chunk = 20, // Min allocation increase
    esz = sizeof(T*), // Element size
  };
  void expand(T**& array,int& size,int index);
public:
  vector(int Owns = 1);
  ~vector();
  T*& operator[](int index);
  int Owns() const { return owns; }
  void Owns(int newOwns) { owns = newOwns; }
};
template<class T>
vector<T>::vector(int Owns)
  : pos(0), pos_sz(0),
    neg(0), neg_sz(0),
    owns(Owns) {}

template<class T>
vector<T>::~vector() {
  if(owns)
    for(int i = 0; i < pos_sz; i++)
      delete pos[i];
  free(pos);
  if(owns)
    for(int j = 0; j < neg_sz; j++)
      delete neg[j];
  free(neg);
}

template<class T>
T*& vector<T>::operator[](int index) {
  if(index < 0) {
    index *= -1;
    if(index >= neg_sz)
      expand(neg, neg_sz, index);
    return neg[index];
  }
  else { // Index >= 0
    if(index >= pos_sz)
      expand(pos, pos_sz, index);
    return pos[index];
  }
}

template<class T> void
vector<T>::expand(T**& array, int& size,
                  int index) {
  const newsize = index + chunk;
  const increment = newsize - size;
  void* v = realloc(array, newsize * esz);
  allegemem(v);
  array = (T**)v;
  memset(&array[size], 0, increment * esz);
  size = index + chunk;
}
#endif // VECTOR_H_
```

[illegible]

[illegible] expand()[illegible] expand()[illegible] T**&[illegible] realloc()[illegible] size[illegible] index [illegible] delete[illegible] size[illegible] int&[illegible]

[illegible] operator[][illegible]

```
//: VECTOR.CPP -- Test "infinite" vector
#include <fstream.h>
#include "..\allege.h"
#include "..\14\vector.h"
#include "..\14\sstring.h"
typedef SString<40> String;

main() {
ifstream source("vector.cpp");
allegefile(source);
const bsz = 255;
```

China-pub.com [illegible]

```
  char buf[bsz];
  vector<String> words;
  int i = 0;
  while(source.getline(buf, bsz)) {
    char* s = strtok(buf, " \t");
    while(s) {
      words[i++] = new String(s);
      s = strtok(0, " \t");
    }
    words[i++] = new String("\n");
  }
  for(int j = 0; words[j]; j++)
    cout << *words[j] << ' ';
}
```

[illegible] C[illegible] strtok()[illegible] String[illegible] words[illegible]

15.6.2 [illegible]

[illegible]

```
//: SET.H -- Each entry in a set is unique
#ifndef SET_H_
#define SET_H_
#include "..\14\vector.h"
#include <assert.h>

template<class Type>
class set {
  vector<Type> elem;
  int max;
  int lastindex; // Efficiency tool
  int within(const Type& e) {
    // Requires Type::operator== :
    for(lastindex = 0; lastindex < max;
        lastindex++)
      if(elem[lastindex]->operator==(e))
        return lastindex;
    return -1;
  }
  // Prevent assignment & copy-construction:
```

```
  void operator=(set&);
  set(set&);
public:
  set() : max(0), lastindex(0) {}
  void add(const Type&);
  int contains(const Type&);
  // Where is it in the set?:
  int index(const Type& e);
  Type& operator[] (int index) {
    // No check for shipping application:
    assert(index >= 0 && index < max);
    return *elem[index];
  }
  int length() const { return max; }
};

template<class Type> void
set<Type>::add(const Type& e) {
  if(!contains(e)) {
    elem[max] = new Type(e);//Copy-constructor
    max++;
  }
}

template<class Type> int
set<Type>::contains(const Type& e) {
  return within(e) != -1;
}

template<class Type> int
set<Type>::index(const Type& e) {
  // Prevent a new search if possible:
  if(elem[lastindex]->operator!=(e)) {
    int ind = within(e);
    assert(ind != -1); // Must know it's inside
  }
  return lastindex;
}
#endif // SET_H_
```

add()□ □□□□□□□□□□□□□□□□□□□□□□□□ contains()□ □□□□□□ □□□□□□□□□□ index()□ □□□□□□□□□□□□□□□□□□ operator[]□ □□□□ length()□ □□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□ index()□ □□□□ contains()□ □□□□□□□□□□□□□□□□□□□□□□ □ within()□ □□□□□□□

[illegible]

```
//: SETTEST.CPP -- Test the "set" class
// Creates a concordance of text words
#include <fstream.h>
#include "..\14\set.h"
#include "..\14\sstring.h"
#include "..\allege.h"
const char* delimiters =
  " \t;()\"<>:{}[]+-=&*#.,/\\"
  "0123456789";

typedef SString<40> String;

main(int argc, char* argv[]) {
  allege(argc == 2, "need file argument");
  ifstream in(argv[1]);
  allegefile(in);
  ofstream out("settest.out");
  set<String> concordance;
  const sz = 255;
  char buf[sz];
  while(in.getline(buf, sz)) {
    // Capture individual words:
    char* s = strtok(buf, delimiters);
    while(s) {
      // Contains 1 entry per unique word:
      concordance.add(s); // Auto type conv.
      s = strtok(0, delimiters);
    }
  }
  for(int i = 0; i < concordance.length(); i++) {
    out << concordance[i] << endl;
  }
}
```

[illegible] strtok()[illegible]

[illegible] add()[illegible] char*[illegible] SString<40>[illegible] char*[illegible] add()[illegible] add()[illegible] add()[illegible] new[illegible]

15.6.3 [illegible]

[illegible]

□□□

□□□ operator=□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ operator[]□□ in_value() □ out_value()□□□□□□□□□□□□□□□□□□□

```
//: ASSOC.H -- Associative array
#ifndef ASSOC_H_
#define ASSOC_H_
#include "..\14\set.h"
#include <assert.h>

template<class In, class Out>
class assoc_array {
  set<In> inVal;
  vector<Out> outVal;
  int max;
  // Prevent assignment & copy-construction:
  void operator=(assoc_array&);
  assoc_array(assoc_array&);
public:
  assoc_array() : max(0) {}
  Out& operator[](const In&);
  int length() const { return max; }
  In& in_value(int i) {
    // No check for shipping application:
    assert(i >= 0 && i < max);
    return inVal[i];
  }
  Out& out_value(int i) {
    assert(i >= 0 && i < max);
    return *outVal[i];
  }
};

template<class In, class Out> Out&
assoc_array<In,Out>::operator[](const In& in){
  if(!inVal.contains(in)) {
    inVal.add(in); // Copy-constructor
    outVal[max] = new Out; // Default constr.
    max++;
  }
  int x = inVal.index(in);
  return *outVal[inVal.index(in)];
}
#endif // ASSOC_H_
```

□□built-in□□□SString□□□□□□□□□□□□□

```
//: ASSOC.CPP -- Test of associative array
#include "..\14\assoc.h"
#include "..\14\sstring.h"
#include "..\14\integer.h"
#include "..\allege.h"
#include <fstream.h>
#include <ctype.h>

main() {
  const char* delimiters =
    " \t;()\"<>:{}[]+-=&*#.,/\\";
  assoc_array<SString<80>, integer> strcount;
  ifstream source("assoc.cpp");
  allegefile(source);
  ofstream out("assoc.out");
  allegefile(out);
  const bsz = 255;
  char buf[bsz];
  while(source.getline(buf, bsz)) {
    char* s = strtok(buf, delimiters);
    while(s) {
      strcount[s]++; // Count word
      s = strtok(0, delimiters);
    }
  }
  for(int i = 0; i < strcount.length(); i++) {
    out << strcount.in_value(i) << " : "
         << strcount.out_value(i) << endl;
  }

  // The "shopping list" problem:
  assoc_array<SString<>, integer> shoplist;
  ifstream list("shoplist.txt");
  allegefile(list);
  ofstream olist("shoplist.out");
  allegefile(olist);
  while(list.getline(buf, bsz)) {
    int i = strlen(buf) - 1; // Last char
    while(isspace(buf[i]))
      i--; // Find nonzero char at end
    while(!isspace(buf[i]))
      i--; // Back up to space or tab
    int count = atoi(&buf[i+1]); // Use value
    while(isspace(buf[i]))
      i--; // Back up to non-whitespace
    buf[i+1] = 0; // Mark end of description
    i = 0;
    while(isspace(buf[i]))
      i++; // Find start of first word
    shoplist[&buf[i]] += count;
  }
  for(int j = 0; j < shoplist.length(); j++) {
    olist << shoplist.in_value(j) << " : "
          << shoplist.out_value(j) << endl;
  }
}
```

□□□□□□□□□□□

strcount[s]++;//count word

□□s□char*□□□□□operator[]□□□□□SString<80>□□□□□□□□□□□□□□□□□□□□□□□ SString<80>(char*)□operator[]□□□□□□□□□□□□□□□□□□□□□□□□□□

operator[]□□□□integer&□assoc_array□□“□□”□□C□□□□isspace()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

shoplist[&buf[i]] +=count;

□□□□□□ char*□□□□□□□□□□SString□□□

15.7 □□□□□

□□□ tstash □□

```
//: SORTED.H -- Template inheritance
#ifndef SORTED_H_
#define SORTED_H_
#include <stdlib.h>
```

```
#include <string.h>
#include <time.h>
#include "..\14\tstash.h"
#include "..\14\set.h"
template<class T>
class sorted :  public tstash<T> {
  void bubblesort();
public:
  int add(T* element) {
    tstash<T>::add(element);
    bubblesort();
    return 0; // Sort moves the element
  }
};
template<class T>
void sorted<T>::bubblesort() {
  for(int i = count(); i > 0; i--)
    for(int j = 1; j < i; j++)
      if(*storage[j-1] > *storage[j]) {
        // Swap the two elements:
        T* t = storage[j-1];
        storage[j-1] = storage[j];
        storage[j] = t;
      }
}

// Quick & dirty sorted set:
template<class T>
class sortedSet : public set<T> {
  sorted<T> Sorted;
public:
  void add(T& e) {
    if(contains(e)) return;
    set<T>::add(e);
    Sorted.add(new T(e));
  }
  T& operator[] (int index) {
    assert(index >= 0 && index < length());
    assert(Sorted[index]);
    return *Sorted[index];
  }
  int length() {
    return Sorted.count();
  }
};
```

```
// Unique random number generator:
template<int upper_bound>
class urand {
  int map[upper_bound];
  int recycle;
public:
  urand(int Recycle = 0);
  int operator()();
};

template<int upper_bound>
urand<upper_bound>::urand(int Recycle = 0)
  : recycle(Recycle) {
  memset(map, 0, upper_bound * sizeof(int));
  // Seed the random number generator:
  time_t t;
  srand((unsigned)time(&t));
}

template<int upper_bound>
int urand<upper_bound>::operator()() {
  if(!memchr(map, 0, upper_bound)) {
    if(recycle)
      memset(map, 0,
             sizeof(map) * sizeof(int));
    else
      return -1; // No more spaces left
  }
  int newval;
  while(map[newval = rand() % upper_bound])
    ; // Until unique value is found
  map[newval]++; // Set flag
  return newval;
}
#endif // SORTED_H_
```

[illegible] operator()[illegible] urand[illegible] map[illegible] map[illegible] map[illegible]

[illegible] sorted[illegible] >[illegible] SString[illegible] integer[illegible] int()[illegible] >

[illegible] integer[illegible]

[illegible] tstash[illegible] storage[illegible] sorted[illegible] tstash[illegible] sorted[illegible] tstash[illegible]

[illegible] sortedSet[illegible] sortedSet[illegible] set[illegible] set[illegible] sorted[illegible] C++[illegible] SORTED.H[illegible]

```
//: SORTED.CPP -- Testing template inheritance
#include "..\14\sorted.h"
#include "..\14\sstring.h"
#include "..\14\integer.h"
typedef SString<40> String;

char* words[] = {
  "is", "running", "big", "dog", "a",
};
const wordsz = sizeof words / sizeof *words;

main() {
  sorted<String> ss;
  for(int i = 0; i < wordsz; i++)
    ss.add(new String(words[i]));
  for(int j = 0; j < ss.count(); j++)
    cout << ss[j]->str() << endl;
  sorted<integer> is;
  urand<47> rand1;
  for(int k = 0; k < 15; k++)
    is.add(new integer(rand1()));
  for(int l = 0; l < is.count(); l++)
    cout << *is[l] << endl;
}
```

[illegible] SString [illegible] interger[illegible]

15.7.1 [illegible]

[illegible] sorted[illegible] add()[illegible]

1) □□□□□□□

2) □□□□□□□□□□□□□

3) □□□□□□□□

□□□□□□□□□□□□□□□□“□□”□□“□□□”□□□C++□□□□□□

15.7.2 □□□□□□

□□□13□□INHSTAK.CPP(□13.5.2)□□□□□□□□□stack□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
//: NOBLOAT.H -- Templatized INHSTAK.CPP
#ifndef NOBLOAT_H_
#define NOBLOAT_H_
#include "..\11\stack11.h"

template<class T>
class nbstack : public stack {
public:
  void push(T* str) {
    stack::push(str);
  }
  T* peek() const {
    return (T*)stack::peek();
  }
  T* pop() {
    return (T*)stack::pop();
  }
  ~nbstack();
};

// Defaults to heap objects & ownership:
template<class T>
nbstack<T>::~nbstack() {
  T* top = pop();
  while(top) {
    delete top;
    top = pop();
  }
}
#endif // NOBLOAT_H_
```

[illegible] ([illegible]) [illegible]

15.8 [illegible]

[illegible] 18[illegible] “[illegible] (RTTI) ” [illegible]

```
//: RECYCLE.CPP -- Containers & polymorphism
#include <fstream.h>
#include <stdlib.h>
#include <time.h>
#include "..\14\tstack.h"
ofstream out("recycle.out");

enum type { Aluminum, Paper, Glass };

class trash {
  float Weight;
public:
  trash(float Wt) : Weight(Wt) {}
  virtual type trashType() const = 0;
  virtual const char* name() const = 0;
  virtual float value() const = 0;
  float weight() const { return Weight; }
  virtual ~trash() {}
};

class aluminum : public trash {
  static float val;
public:
  aluminum(float Wt) : trash(Wt) {}
  type trashType() const { return Aluminum; }
  virtual const char* name() const {
    return "aluminum";
  }
  float value() const { return val; }
  static void value(int newval) {
    val = newval;
  }
};

float aluminum::val = 1.67;

class paper : public trash {
  static float val;
public:
  paper(float Wt) : trash(Wt) {}
  type trashType() const { return Paper; }
  virtual const char* name() const {
    return "paper";
  }
  float value() const { return val; }
  static void value(int newval) {
    val = newval;
  }
};

float paper::val = 0.10;

class glass : public trash {
  static float val;
public:
  glass(float Wt) : trash(Wt) {}
  type trashType() const { return Glass; }
  virtual const char* name() const {
    return "glass";
  }
  float value() const { return val; }
  static void value(int newval) {
    val = newval;
  }
};

float glass::val = 0.23;

// Sums up the value of the trash in a bin:
void SumValue(const tstack<trash>& bin,ostream& os){
  tstackIterator<trash> tally(bin);
  float val = 0;
  while(tally) {
    val += tally->weight() * tally->value();
    os << "weight of " << tally->name()
        << " = " << tally->weight() << endl;
```

```
      tally++;
    }
    os << "Total value = " << val << endl;
  }

  main() {
    // Seed the random number generator
    time_t t;
  srand((unsigned)time(&t));

  tstack<trash> bin; // Default to ownership
  // Fill up the trash bin:
  for(int i = 0; i < 30; i++)
    switch(rand() % 3) {
      case 0 :
        bin.push(new aluminum(rand() % 100));
        break;
      case 1 :
        bin.push(new paper(rand() % 100));
        break;
      case 2 :
        bin.push(new glass(rand() % 100));
        break;
    }
  // Bins to sort into:
  tstack<trash> glassbin(0); // No ownership
  tstack<trash> paperbin(0);
  tstack<trash> ALbin(0);
  tstackIterator<trash> sorter(bin);
  // Sort the trash:
  // (RTTI offers a nicer solution)
  while(sorter) {
    // Smart pointer call:
    switch(sorter->trashType()) {
      case Aluminum:
        ALbin.push(sorter.current());
        break;
      case Paper:
        paperbin.push(sorter.current());
        break;
      case Glass:
        glassbin.push(sorter.current());
        break;
    }
    sorter++;
    }
    SumValue(ALbin, out);
    SumValue(paperbin, out);
    SumValue(glassbin, out);
    SumValue(bin, out);
  }
```

[illegible] tstack [illegible] trash[illegible] trash[illegible] tstack[illegible] trash[illegible] push()[illegible] trash[illegible] tally [illegible] sorter[illegible]

trash[illegible] bin[illegible]

[illegible] “ [illegible] ” [illegible]

15.9 [illegible]

[illegible] “ [illegible] ” [illegible] “ [illegible] ” [illegible]

[illegible]

[illegible]

[illegible] C++[illegible] STL[illegible] standard template library[illegible] STL[illegible] A[illegible]

[illegible] STL[illegible]

[illegible] STL[illegible] (associative container)[illegible] STL[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible] STL [illegible]

[illegible]

[illegible] STL [illegible] “ [illegible] ” [illegible] “ [illegible] ” [illegible] STL [illegible]

[illegible] STL [illegible] STL [illegible]

[illegible] “ [illegible] ” “ [illegible] ” [illegible]

[illegible] STL [illegible]

• [illegible]

[illegible] STL [illegible]

15.10 [illegible]

[illegible] C++ [illegible]

[illegible] [1] [illegible]

15.10.1 [illegible]

[illegible] malloc() [illegible] calloc() [illegible] realloc() [illegible]

---

[1] [illegible] C++Inside & out(Osborne/McGraw-Hill, 1993)[illegible]

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ realloc())□□□ getmem() □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ getmem()□□

```
//: GETMEM.H -- Function template for memory
#ifndef GETMEM_H_
#define GETMEM_H_
#include <stdlib.h>
#include <string.h>
#include "..\allege.h"

template<class T>
void getmem(T*& oldmem, int elems) {
  typedef int cntr; // Type of element counter
  const int csz = sizeof(cntr); // And size
  const int Tsz = sizeof(T);
  if(elems == 0) {
    free(&(((cntr*)oldmem)[-1]));
    return;
  }
  T* p = oldmem;
  cntr oldcount = 0;
  if(p) { // Previously allocated memory
    ((cntr*)p)--; // Back up by one cntr
    oldcount = *(cntr*)p; // Previous # elems
  }
  T* m = (T*)realloc(p, elems * Tsz + csz);
  allegemem(m);
  *((cntr*)m) = elems; // Keep track of count
  const cntr increment = elems - oldcount;
  if(increment > 0) {
    // Starting address of data:
    long startadr = (long)&(m[oldcount]);
    startadr += csz;
    // Zero the additional new memory:
    memset((void*)startadr, 0, increment * Tsz);
  }
  // Return the address beyond the count:
  oldmem = (T*)&(((cntr*)m)[1]);
}

template<class T>
inline void freemem(T * m) { getmem(m, 0); }

#endif // GETMEM_H_
```

[illegible] typedef cntr [illegible]

[illegible] oldmem[illegible] oldmem[illegible] getmem()[illegible] getmem()[illegible]

[illegible] freemem[illegible]

[illegible] getmem()[illegible] oldmem[illegible] free()[illegible] getmem()[illegible] cntr[illegible] oldmem[illegible] T*[illegible] cntr*[illegible] free()[illegible]

```
free(&(((cntr*) oldmem )[-1]));
```

[illegible] getmem()[illegible] cntr[illegible] realloc()[illegible] memset()[illegible] oldmem[illegible]

```
oldmem=(T*)&(((cntr*)m)[1]);
```

[illegible] oldmem[illegible] getmem()[illegible]

[illegible] getmem()[illegible]

```
//: GETMEM.CPP -- Test memory function template
#include "..\14\getmem.h"
#include <iostream.h>

main() {
  int* p = 0;
  getmem(p, 10);
  for(int i = 0; i < 10; i++) {
    cout << p[i] << ' ';
    p[i] = i;
  }
  cout << '\n';
  getmem(p, 20);
  for(int j = 0; j < 20; j++) {
    cout << p[j] << ' ';
    p[j] = j;
  }
  cout << '\n';
  getmem(p, 25);
for(int k = 0; k < 25; k++)
  cout << p[k] << ' ';
freemem(p);
cout << '\n';

float* f = 0;
getmem(f, 3);
for(int u = 0; u < 3; u++) {
    cout << f[u] << ' ';
    f[u] = u + 3.14159;
  }
  cout << '\n';
  getmem(f, 6);
  for(int v = 0; v < 6; v++)
    cout << f[v] << ' ';
  freemem(f);
}
```

□□□□□ getmem()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□ getmem()□□ void* &□□ oldmem□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ void*□□

15.10.2 □ tstack□ □□□

□□□□□□□□□□ tstack□□□□□□□□□□□□□□□□□□□ tstack□□□□□□□□□□□□□□□□□□□□□□□ tstack□□□□□□□□□□□□□□□□□□□□

```
//: APPLIST.CPP -- Apply a function to a tstack
#include "..\14\tstack.h"
#include <iostream.h>

// 0 arguments, any type of return value:
template<class T, class R>
void applist(tstack<T>& tl, R(T::*f)()) {
  tstackIterator<T> it(tl);
  while(it) {
    (it.current()->*f)();
    it++;
  }
}

// 1 argument, any type of return value:
template<class T, class R, class A>
void applist(tstack<T>& tl, R(T::*f)(A), A a) {
  tstackIterator<T> it(tl);
  while(it) {
    (it.current()->*f)(a);
    it++;
  }
}

// 2 arguments, any type of return value:
template<class T, class R, class A1, class A2>
void applist(tstack<T>& tl, R(T::*f)(A1, A2),
    A1 a1, A2 a2) {
  tstackIterator<T> it(tl);
  while(it) {
    (it.current()->*f)(a1, a2);
    it++;
  }
}

// Etc., to handle maximum probable arguments

class gromit { // The techno-dog
  int arf;
public:
  gromit(int Arf = 1) : arf(Arf + 1) {}
  void speak(int) {
    for(int i = 0; i < arf; i++)
      cout << "arf! ";
    cout << endl;
  }
  char eat(float) {
    cout << "chomp!" << endl;
    return 'z';
  }
  int sleep(char, double) {
    cout << "zzz..." << endl;
    return 0;
  }
  void sit(void) {}
};

main() {
  tstack<gromit> dogs;
  for(int i = 0; i < 5; i++)
    dogs.push(new gromit(i));
  applist(dogs, &gromit::speak, 1);
  applist(dogs, &gromit::eat, 2.0f);
  applist(dogs, &gromit::sleep, 'z', 3.0);
  applist(dogs, &gromit::sit);
}
```

applist()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□applist()□□10□□□□□□□

□□□□□□□□□□□applist()□□applist()□□□gromit□□□□□□□□□□□□□[1]□□□□□□□“□□□”□□□□□□□□□□□□□□□□□□□□□□□□□

□□applist()□□

□□□□□□□□□□□□□□□□tstack□□□□□□□□□□□tstack□□□□□□□□□□□□□□□□□□□□□□□

15.10.3 □□□□□□

□applist()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□,□□□□□□□□□□□□□□□□□□□□□□□□□□

```
dogs.applist(&gromit::sit);
```

□□□□□□□□□□□□□□□□□2□□□□□[2]□

15.11 □□□□

□□□

```
template class bobbin<thread>;
template void sort<char>(char *[]);
```

□□SORTED.CPP□□□□□□□□□15.7□□□□□□□□□□□□□□□□□□□

```
//: GENERATE.CPP -- Explicit instantiation
#include "..\14\sorted.h"
#include "..\14\integer.h"
// Explicit instantiation:
template class sorted<integer>;

main() {
  sorted<integer> is;
  urand<47> rand1;
```

---

[1] □□□Nick Park□□□□□□□□□□□□□□

[2] □□□□□□□□□□□□□□□□□□□□□□□□□

```
  for(int k = 0; k < 15; k++)
    is.add(new integer(rand1()));
  for(int l = 0; l < is.count(); l++)
    cout << *is[l] << endl;
}
```

□□□

•□□□□□

sorted□□□□□□□□□□□□□□□□□□□□□□ char*□□char*□ sorted□□□

```
//: SPECIAL.CPP -- Template specialization
// A special sort for char*
#include "..\14\sorted.h"
#include <iostream.h>
class sorted<char> :  public tstash<char> {
  void bubblesort();
public:
  int add(char* element) {
    tstash<char>::add(element);
    bubblesort();
    return 0; // Sort moves the element
  }
};

void sorted<char>::bubblesort() {
  for(int i = count(); i > 0; i--)
    for(int j = 1; j < i; j++)
      if(strcmp(storage[j], storage[j-1]) < 0) {
        // Swap the two elements:
        char* t = storage[j-1];
        storage[j-1] = storage[j];
        storage[j] = t;
      }
}

char* words[] = {
  "is", "running", "big", "dog", "a",
};
const wsz = sizeof words/sizeof *words;

main() {
  sorted<char> sc;
  for(int k = 0; k < wsz; k++)
```

```
    sc.add(words[k]);
  for(int l = 0; l < sc.count(); l++)
    cout << sc[l] << endl;
}
```

[illegible] bubblesort()[illegible] strcmp()[illegible] >[illegible]

15.12 [illegible]

[illegible] OOP[illegible] C[illegible]

[illegible]

[illegible] [1][illegible] C++[illegible]

[illegible] 16[illegible] 12[illegible]

15.13 [illegible]

1. [illegible] 14[illegible] tstack [illegible] tstackIterator[illegible] shape[illegible] tstack[illegible] shape[illegible]
2. [illegible] 14[illegible] SSHAPE2.CPP[illegible] tstack[illegible]
3. [illegible] RECYCLE.CPP[illegible] tstash[illegible] tstack[illegible]
4. [illegible] SETTEST.CPP[illegible] sortedSet[illegible] set.[illegible]
5. [illegible] tstash [illegible] APPLIST.CPP[illegible]
6. [illegible] TSTACK.H[illegible] APPLIST.CPP[illegible] tstack[illegible]
7. [illegible] tstack[illegible] add()[illegible] add()[illegible]
8. [illegible] tstack[illegible])[illegible]
9. [illegible] SORTED.CPP[illegible] urand[illegible] SORTED.CPP[illegible])[illegible]
10. [illegible] GETMEM.H[illegible] typedef cntr[illegible]
11. [illegible] SString[illegible]

---

[1] C++[illegible]

[illegible]

16

MI multiple inheritance)

MI MI

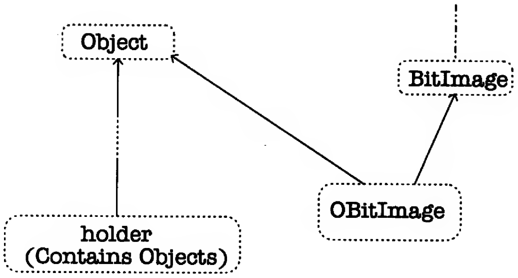
16.1

C++Smalltalk SmalltalkSmalltalk“   ” ,C++“   ”C++ CSmalltalkObjectSmalltalkSmalltalkSmalltalkSmalltalk class

SmalltalkObjectC++C++

MIC++AT&T cfront2.0 MIMIMI“ ”

MIvoid*pstash stackSmalltalkObjectsObjectSmalltalkSmalltalkObjectObjects


Object
BitImage
OBitImage
holder
(Contains Objects)

 16-1

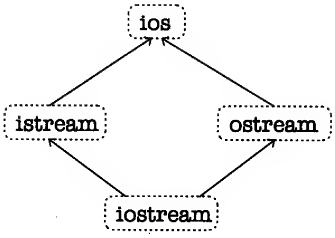
C++AholderBBitImageBitImageholderBitImageholder

MI15MI

    MI

   MI

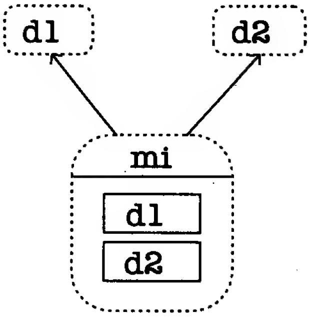
    MI


ios
istream
ostream
iostream

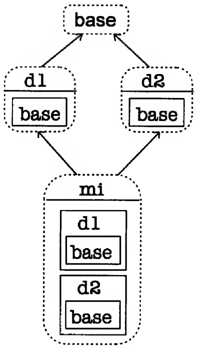
 16-2

16.2

  d1 d2   mi mi   d1   d2   mi


d1
d2
mi
d1
d2

 16-3


base
d1
base
d2
base
mi
d1
base
d2
base

 16-4

     d1 d2   base

       d1 d2   base   mi

              “ ”

16.3

            mi   base

base   ?

```
//: MULTIPL1.CPP -- MI & ambiguity
#include <iostream.h>
#include "..\14\tstash.h"

class base {
public:
  virtual char* vf() const = 0;
};

class d1 : public base {
public:
  char* vf() const { return "d1"; }
};

class d2 : public base {
public:
  char* vf() const { return "d2"; }
};

// Causes error: ambiguous override of vf():
//! class mi : public d1, public d2 {};

main() {
  tstash<base> b;
  b.add(new d1);
  b.add(new d2);
  // Cannot upcast: which subobject?:
//!  b.add(new mi);
  for(int i = 0; i < b.count(); i++)
    cout << b[i]->vf() << endl;
}
```

□□□□□□□□□□□□□□ d1□ d2□ □□ vf()□ □□□□□□□□□□□□□□□□ mi□ □□□□□□ b[]□ □□□□□□□□□□□ new mi □□□□□□□ base*□ □□□□□□ □□□□□□□□ d1□ □□□ base□ □ d2□ □□□ base□ □□□□□□□□□□□□□□□ □□

16.4 □□□

□□□□□□□□□□□□□□ mi□ □□□ vf()□ □□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□ virtual□ □□□□□□□□□ virtual □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ pointer magic□ □□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□

```
//: MULTIPL2.CPP -- Virtual Base Classes
#include <iostream.h>
#include "..\14\tstash.h"

class base {
public:
  virtual char* vf() const = 0;
};

class d1 : virtual public base {
public:
  char* vf() const { return "d1"; }
};

class d2 : virtual public base {
public:
  char* vf() const { return "d2"; }
};
// MUST explicitly disambiguate vf():
class mi : public d1, public d2 {
public:
  char* vf() const { return d1::vf();}
};

main() {
  tstash<base> b;
  b.add(new d1);
  b.add(new d2);
  b.add(new mi); // OK
  for(int i = 0; i < b.count(); i++)
    cout << b[i]->vf() << endl;
}
```

[illegible] mi[illegible] vf()[illegible]

16.4.1 “[illegible]”[illegible]

[illegible] most-derived[illegible]

[illegible] base[illegible] d1[illegible] d1[illegible] mi[illegible] mi[illegible]

[illegible]

```
//: MULTIPL3.CPP -- Virtual base initialization
// Virtual base classes must always be
// Intialized by the "most-derived" class
#include <iostream.h>
#include "..\14\tstash.h"
```

```
class base {
public:
  base(int) {}
  virtual char* vf() const = 0;
};

class d1 : virtual public base {
public:
  d1() : base(1) {}
  char* vf() const { return "d1"; }
};

class d2 : virtual public base {
public:
  d2() : base(2) {}
  char* vf() const { return "d2"; }
};

class mi : public d1, public d2 {
public:
  mi() : base(3) {}
  char* vf() const {
    return d1::vf(); // MUST disambiguate
  }
};

class x : public mi {
public:
  // You must ALWAYS init the virtual base:
  x() : base(4) {}
};

main() {
  tstash<base> b;
  b.add(new d1);
  b.add(new d2);
  b.add(new mi); // OK
  b.add(new x);
  for(int i = 0; i < b.count(); i++)
    cout << b[i]->vf() << endl;
}
```

d1□d2□□□□□□□□□□□□□base□□□□□□□□□□□□□□□mi□x□□□□□□□□□□base□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□d1□□□base□□□□d2□□

16.4.2 □□□□□□□□□□□“□□”

□□□,□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
//: MULTIPL4.CPP -- "Tying off" virtual bases
// so you don't have to worry about them
// in derived classes
#include <iostream.h>
#include "..\14\tstash.h"
class base {
public:
  // Default constructor removes responsibility:
  base(int = 0) {}
  virtual char* vf() const = 0;
};

class d1 : virtual public base {
public:
  d1() : base(1) {}
  char* vf() const { return "d1"; }
};

class d2 : virtual public base {
public:
  d2() : base(2) {}
  char* vf() const { return "d2"; }
};

class mi : public d1, public d2 {
public:
  mi() {} // Calls default constructor for base
  char* vf() const {
    return d1::vf(); // MUST disambiguate
  }
};

class x : public mi {
public:
  x() {} // Calls default constructor for base
};

main() {
  tstash<base> b;
  b.add(new d1);
  b.add(new d2);
  b.add(new mi); // OK
  b.add(new x);
  for(int i = 0; i < b.count(); i++)
    cout << b[i]->vf() << endl;
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

16.5 □□

□□“□□□□□ pointer magic□”□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
//: OVERHEAD.CPP -- Virtual base class overhead
#include <fstream.h>
ofstream out("overhead.out");

class base {
public:
  virtual void f() const {};
};

class nonvirtual_inheritance
  : public base {};

class virtual_inheritance
  : virtual public base {};

class virtual_inheritance2
  : virtual public base {};

class mi
  : public virtual_inheritance,
    public virtual_inheritance2 {};
#define WRITE(arg) \
out << #arg << " = " << arg << endl;

main() {
  base b;
  WRITE(sizeof(b));
  nonvirtual_inheritance nonv_inheritance;
  WRITE(sizeof(nonv_inheritance));
  virtual_inheritance v_inheritance;
  WRITE(sizeof(v_inheritance));
  mi MI;
  WRITE(sizeof(MI));
}
```

□□□□□□□□□□□□□□□“ □□□□□ core size□”□□□

sizeof(b)=2
sizeof(nonv_inheritance)=2
sizeof (v_inheritance)=6
sizeof(MI)=12

□□□□□b□ nonv_inheritance□□□□□□□□□□□□□□□□□□□□□ VPTR□□ VPTR□□

□□□[1]□□

16.6 □□□□

□□□ this□□ this□□□□□□□□□□□□□□□□□□

```
//: MITHIS.CPP -- MI and the "this" pointer
#include <fstream.h>
ofstream out("mithis.out");

class base1 {
  char c[0x10];
public:
  void printthis1() {
    out << "base1 this = " << this << endl;
  }
};

class base2 {
  char c[0x10];
public:
  void printthis2() {
    out << "base2 this = " << this << endl;
  }
};

class member1 {
  char c[0x10];
public:
```

---

[1] □ Jan Gray C++Under the Hood, a chapter in Black Belt C++(edited by Bruce Eckel, M& T press, 1995)□

```
  void printthism1() {
    out << "member1 this = " << this << endl;
  }
};

class member2 {
  char c[0x10];
public:
  void printthism2() {
    out << "member2 this = " << this << endl;
  }
};

class mi : public base1, public base2 {
  member1 m1;
  member2 m2;
public:
  void printthis() {
    out << "mi this = " << this << endl;
    printthis1();
    printthis2();
    m1.printthism1();
    m2.printthism2();
  }
};

main() {
  mi MI;
  out << "sizeof(mi) = "
    << hex << sizeof(mi) << " hex" << endl;
  MI.printthis();
  // A second demonstration:
  base1* b1 = &MI; // Upcast
  base2* b2 = &MI; // Upcast
  out << "base 1 pointer = " << b1 << endl;
  out << "base 2 pointer = " << b2 << endl;
}
```

□□ this□□□□□□□□□□□□□□□□□□□□□□□□ mi□□□ this□□□□□□ mi□□□□□□□□□□□□□□□□□□

```
sizeof(mi)=40 hex
mi this=0x223e
base1 this=0x223e
```

```
base2 this=0x224e
member1 this=0x225e
member2 this=0x226e
base 1 pointer=0x223e
base 2 pointer=0x224e
```

[illegible] C++[illegible]

[illegible] base1[illegible] base2[illegible] this[illegible]

[illegible]

[illegible]

[illegible] “ [illegible] ”[illegible]

[illegible]

1) [illegible]

2) [illegible]

3) [illegible]

4) [illegible]

[illegible] serialization[illegible] deserialization[illegible] serialize() [illegible]

1. [illegible] MI[illegible]

[illegible] persisence[illegible]

```
//: PERSIST1.CPP -- Simple persistence with MI
#include <fstream.h>

class persistent {
  int objSize; // Size of stored object
public:
  persistent(int sz) : objSize(sz) {}
  void write(ostream& out) const {
    out.write((char*)this, objSize);
  }
  void read(istream& in) {
    in.read((char*)this, objSize);
  }
};
class data {
   float f[3];
public:
   data(float f0 = 0.0, float f1 = 0.0,
     float f2 = 0.0) {
     f[0] = f0;
     f[1] = f1;
     f[2] = f2;
   }
   void print(const char* msg = "") const {
     if(*msg) cout << msg << "   ";
     for(int i = 0; i < 3; i++)
       cout << "f[" << i << "] = "
            << f[i] << endl;
   }
};

class wdata1 : public persistent, public data {
public:
   wdata1(float f0 = 0.0, float f1 = 0.0,
     float f2 = 0.0) : data(f0, f1, f2),
     persistent(sizeof(wdata1)) {}
};

class wdata2 : public data, public persistent {
public:
   wdata2(float f0 = 0.0, float f1 = 0.0,
     float f2 = 0.0) : data(f0, f1, f2),
     persistent(sizeof(wdata2)) {}
};

main() {
   {
     ofstream f1("f1.dat"), f2("f2.dat");
     wdata1 d1(1.1, 2.2, 3.3);
     wdata2 d2(4.4, 5.5, 6.6);
     d1.print("d1 before storage");
```

```
      d2.print("d2 before storage");
      d1.write(f1);
      d2.write(f2);
    } // Closes files
    ifstream f1("f1.dat"), f2("f2.dat");
    wdata1 d1;
    wdata2 d2;
    d1.read(f1);
    d2.read(f2);
    d1.print("d1 after storage");
    d2.print("d2 after storage");
  }
```

[illegible] persistent::read() [illegible] persistent::write() [illegible] this [illegible] read() [illegible] write() [illegible] IO [illegible] write() [illegible]

[illegible] 18 [illegible] “[illegible]” [illegible] sizeof [illegible]

[illegible] data [illegible] VPTR [illegible] wdata1 [illegible] main() [illegible] F1.DAT [illegible] persistent [illegible] wdata2 [illegible] persistent [illegible] this [illegible]

[illegible] this [illegible] this [illegible] 18 [illegible] “[illegible]” [illegible]

2. [illegible]

[illegible][1] [illegible]

```
//: PERSIST2.CPP -- Improved MI persistence
#include <fstream.h>
#include <string.h>

class persistent {
public:
```

---

[1] [illegible]

```
  virtual void write(ostream& out) const = 0;
  virtual void read(istream& in) = 0;
};
class data {
protected:
  float f[3];
public:
  data(float f0 = 0.0, float f1 = 0.0,
    float f2 = 0.0) {
    f[0] = f0;
    f[1] = f1;
    f[2] = f2;
  }
  void print(const char* msg = "") const {
    if(*msg) cout << msg << endl;
    for(int i = 0; i < 3; i++)
      cout << "f[" << i << "] = "
           << f[i] << endl;
  }
};

class wdata1 : public persistent, public data {
public:
  wdata1(float f0 = 0.0, float f1 = 0.0,
    float f2 = 0.0) : data(f0, f1, f2) {}
  void write(ostream& out) const {
    out << f[0] << " " << f[1] << " " << f[2];
  }
  void read(istream& in) {
    in >> f[0] >> f[1] >> f[2];
  }
};

class wdata2 : public data, public persistent {
public:
  wdata2(float f0 = 0.0, float f1 = 0.0,
    float f2 = 0.0) : data(f0, f1, f2) {}
  void write(ostream& out) const {
    out << f[0] << " " << f[1] << " " << f[2];
  }
  void read(istream& in) {
    in >> f[0] >> f[1] >> f[2];
  }
};

class conglomerate : public data,
  public persistent {
    char* name; // Contains a pointer
    wdata1 d1;
    wdata2 d2;
  public:
    conglomerate(const char* nm = "",
      float f0 = 0.0, float f1 = 0.0,
      float f2 = 0.0, float f3 = 0.0,
      float f4 = 0.0, float f5 = 0.0,
      float f6 = 0.0, float f7 = 0.0,
      float f8= 0.0) : data(f0, f1, f2),
      d1(f3, f4, f5), d2(f6, f7, f8) {
      name = new char[strlen(nm) + 1];
      strcpy(name, nm);
    }
    void write(ostream& out) const {
      int i = strlen(name) + 1;
      out << i << " "; // Store size of string
      out << name << endl;
      d1.write(out);
      d2.write(out);
      out << f[0] << " " << f[1] << " " << f[2];
    }
    // Must read in reverse order as write:
    void read(istream& in) {
      delete []name; // Remove old storage
      int i;
      in >> i >> ws; // Get int, strip whitespace
      name = new char[i];
      in.getline(name, i);
    d1.read(in);
    d2.read(in);
    in >> f[0] >> f[1] >> f[2];
  }
  void print() const {
    data::print(name);
    d1.print();
    d2.print();
  }
};

main() {
  {
    ofstream data("data.dat");
    conglomerate C("This is conglomerate C",
      1.1, 2.2, 3.3, 4.4, 5.5,
    6.6, 7.7, 8.8, 9.9);
  cout << "C before storage" << endl;
  C.print();
  C.write(data);
} // Closes file
ifstream data("data.dat");
conglomerate C;
C.read(data);
cout << "after storage: " << endl;
C.print();
}
```

□□persistent□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□data□□data□□□protected□

□wdata1□wdata2□□□□□□IO□□□□□□□□□□□□□□□□□□□□□□data□□□□□□□□write()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□conglomerate□□□□□data□□□□□□wdata1□wdata2□□□□□□□□□□□□□□□□□□persistent□□□□□□□□□□VPTR□□□□□□□□□□□□□□□□□□□□□□

□□□write()□read()□□□□□read()□□□□□□□write()□□□□□□□□□□□□□□□□□read()□□□□□□write()□□□□□□□□□□□□□□□□□□char*□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□read()□□□□□□□□□□□□

□□□□□□□□□read()□write()□□□□□□□□□□□□□□□□□□read() □write()□□□□□□□□□□

□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“□□□”□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C++□□□□□□□□□□□□□□□□□□□□□

16.7 □□MI

□PERSIST2.CPP□□□data□□□□□□□□□□□read() □write()□□□□data□wdata1□wdata2□□□□□□□□□□□□□□□MI□□

1) □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

2) □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

[illegible]“[illegible]”[illegible] MI[illegible]

[illegible]

16.8 [illegible]

[illegible]

[illegible]

[illegible]

```
//: VENDOR.H -- Vendor-suppplied class header
// You only get this & the compiled VENDOR.OBJ
#ifndef VENDOR_H_
#define VENDOR_H_

class vendor {
public:
  virtual void v() const;
  void f() const;
  ~vendor();
};

class vendor1 : public vendor {
public:
  void v() const;
  void f() const;
  ~vendor1();
};

void A(const vendor&);
void B(const vendor&);
// Etc.
#endif // VENDOR_H_
```

[illegible] A()[illegible] B()[illegible]

```
//: VENDOR.CPP -- Implementation of VENDOR.H
// This is compiled and unavailable to you
```

```
#include <fstream.h>
#include "..\15\vendor.h"

extern ofstream out; // For trace info

void vendor::v() const {
  out << "vendor::v()\n";
}

void vendor::f() const {
  out << "vendor::f()\n";
}

vendor::~vendor() {
  out << "~vendor()\n";
}

void vendor1::v() const {
  out << "vendor1::v()\n";
}
void vendor1::f() const {
  out << "vendor1::f()\n";
}

vendor1::~vendor1() {
  out << "~vendor1()\n";
}

void A(const vendor& V) {
  // ...
  V.v();
  V.f();
  //...
}

void B(const vendor& V) {
  // ...
  V.v();
  V.f();
  //...
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ VENDOR.OBJ □ VENDOR.LIB□□□□□□□□□□□□□□□

□□

f()
A()B()

```
//: PASTE.CPP -- Fixing a mess with MI
#include <fstream.h>
#include "..\15\vendor.h"

ofstream out("paste.out");

class mybase { // Repair vendor interface
public:
  virtual void v() const = 0;
  virtual void f() const = 0;
  // New interface function:
  virtual void g() const = 0;
  virtual ~mybase() { out << "~mybase()\n"; }
};

class paste1 : public mybase, public vendor1 {
public:
  void v() const {
    out << "paste1::v()\n";
    vendor1::v();
  }
  void f() const {
    out << "paste1::f()\n";
    vendor1::f();
  }
  void g() const {
    out << "paste1::g()\n";
  }
  ~paste1() { out << "~paste1()\n"; }
};

main() {
  paste1& p1p = *new paste1;
mybase& mp = p1p; // Upcast
out << "calling f()\n";
mp.f();  // Right behavior
out << "calling g()\n";
mp.g(); // New behavior
out << "calling A(p1p)\n";
A(p1p); // Same old behavior
```

```
  out << "calling B(p1p)\n";
  B(p1p);  // Same old behavior
  out << "delete mp\n";
  // Deleting a reference to a heap object:
  delete &mp; // Right behavior
}
```

[illegible] mybase([illegible] MI)[illegible] ,f()[illegible] g()[illegible] MI[illegible] paste1::v() [illegible] paste1::f()[illegible] main()[illegible] mybase[illegible]

mybase * mp=plp;// upcast

[illegible] delete[illegible] mp[illegible] g()[illegible] mp[illegible]

```
calling f()
paste1::f()
vendor1::f()
calling g()
paste1::g()
calling A(p1p)
paste1::v()
vendor1::v()
vendor::f()
calling B(p1p)
paste1::v()
vendor1::v()
vendor::f()
delete mp
~paste1()
~vendor1()
~vendor()
~mybase()
```

[illegible] A()[illegible] B()[illegible] v()[illegible]

[illegible]

16.9 [illegible]

[illegible] C++[illegible] OOP[illegible] OOP[illegible] C++[illegible]“[illegible]”[illegible] Smalltalk[illegible] C++[illegible]

[illegible]“[illegible]”[illegible] MI[illegible]“[illegible]”[illegible]

[illegible] Zack Urlocker[illegible]“[illegible] goto”[illegible] goto[illegible] MI[illegible] C++[illegible]“[illegible]”[illegible] Occam[illegible] Occam’ s Razor[illegible]“[illegible]”[illegible]

16.10 [illegible]

1. [illegible] MI[illegible] int[illegible] void[illegible] f()[illegible] X[illegible] X[illegible] Y[illegible] Z[illegible] Y[illegible] Z[illegible] int[illegible] Y[illegible] Z[illegible] A[illegible] A[illegible] f()[illegible]

2. [illegible] X[illegible] px[illegible] A[illegible] px[illegible] X[illegible] A[illegible] X[illegible]

3. [illegible] f()[illegible] px[illegible] f()[illegible]

□17□ □□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□

□□,□□□□□□□□□□□□□□□□□□□□□□□ ,□□ printf( )□□

C□□

□□□□□C++□□□□□□□□□□□□□□□□□□□□□□□□□

1) □□□□□□□□□□□□□□□□□□□□□□□□□“□□”□□□

2) □□□

□□□□□□□□□□C□□□□□□□□□□□□□□□□□□C□□□□□□□□□□□□□□□C++□□□□□□□try□ throw□ catch□□□□□□□C++□□□□□□□□□

17.1 C□□□□□□□

□□□□8□□□□□C□□□□assert( )□□□□□□□□□□□□8□□□assert( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□#define NDEBUG□□□□□□□□□□□□□□□□□□□□□□□□assert( )□ allege( )□□□8□□□□□□□□□□□□□□□□□□□“□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□”□ allege( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□

□□C□□□□□□□□□□□□□□□

1) □□C□□□errno( )□ perror( )□□

2) [illegible] C[illegible] signal()[illegible] raise()[illegible]

3) [illegible] C[illegible] setjmp( ) [illegible] longjmp( )[illegible] setjmp( ) [illegible] longjmp( )[illegible] setjmp( ) [illegible]

[illegible] C++[illegible] C[illegible] setjmp/longjmp[illegible] setjmp/longjmp[illegible]

```
//: NONLOCAL.CPP -- setjmp & longjmp
#include <iostream.h>
#include <setjmp.h>

class rainbow {
public:
  rainbow() { cout << "rainbow()" << endl; }
  ~rainbow() { cout << "~rainbow()" << endl; }
};

jmp_buf kansas;

void oz() {
  rainbow RB;
  for(int i = 0; i < 3; i++)
    cout << "there's no place like home\n";
  longjmp(kansas, 47);
}

main() {
  if(setjmp(kansas) == 0) {
    cout << "tornado, witch, munchkins...\n";
    oz();
  } else {
    cout << "Auntie Em! "
         << "I had the strangest dream..."
         << endl;
  }
}
```

setjmp()[illegible] jmp_buf[illegible] jmp_buf[illegible] longjmp()[illegible] setjmp()[illegible] setjmp()[illegible] longjmp()[illegible] longjmp()[illegible]

[illegible] jmp_buf[illegible] goto[illegible] setjmp/longjmp[illegible]

[illegible] C++[illegible] longjmp()[illegible][1][illegible] C++[illegible]

17.2 [illegible]

[illegible]

```
throw myerror ("something bad happened");
```

myerror[illegible]

[illegible] throw[illegible] throw[illegible] throw[illegible]

[illegible]

[illegible]

17.3 [illegible]

[illegible] C++[illegible]

17.3.1 try[illegible]

[illegible] try[illegible]

```
try {
    // code that may generate exceptions
}
```

---

[1] [illegible]——[illegible] C++[illegible] longjmp( )[illegible]

[illegible]

17.3.2 [illegible]

[illegible] try[illegible] catch[illegible]

```
try {
// code that may generate exceptions
} catch(type1 id1) {
  // handle exceptions of type1
} catch(type2 id2) {
  // handle exceptions of type2
}
// etc...
```

[illegible] catch[illegible] id1[illegible] id2 [illegible]

[illegible] catch[illegible] catch[illegible] switch[illegible] case[illegible] break[illegible]

[illegible]

• [illegible]

[illegible] C++[illegible]

[illegible] ——C ++[illegible] while[illegible]

[illegible]

17.3.3 □□□□□□

□□□C++□□□

□□□□□□□□□□□□□□ throw□□□□□□□□□□□□□□□□□□ throw□□□□□□□□□□□□□□□□□□□□□□□□□□□□

void f ( ) throw (toobig, toosmall, divzero)□

□□□□□□□□

void f ( );

□□□□□□□□□□□□□□□□

□□□□

void f ( ) throw ( );

□□□□□□□□□□□□□□

□□

1. unexpected( )

□□ unexpected( )□

2. set_unexpected( )

unexpected( )□□□ set_new_handler( )□□□ set_unexpected( )□□□□ set_unexpected( ) □□□□□□□□□□□□□□□□□□□□□ void□□□□□□□ unexpected□□□□□□□□□□□□□ unexpected( )□□□□□□□□□□□□□□□□□□□□ set_unexpected( )□□□□□□□□□□□ EXCEPT.H□□□□□□□□□□□□□□□□□□□□□□□□□

```
//: EXCEPT.CPP -- Basic exceptions
// Exception specifications & unexpected()
#include <except.h>
#include <iostream.h>
#include <stdlib.h>
#include <string.h>

class up {};
class fit {};
void g();

void f(int i) throw (up, fit) {
  switch(i) {
    case 1: throw up();
    case 2: throw fit();
  }
  g();
}

// void g() {} // version 1
void g() { throw 47; } // Version 2
// (can throw built-in types)

void my_unexpected() {
  cout << "unexpected exception thrown";
  exit(1);
}

main() {
  set_unexpected(my_unexpected);
  // (ignores return value)
  for(int i = 1; i <=3; i++)
    try {
      f(i);
    } catch(up) {
      cout << "up caught" << endl;
    } catch(fit) {
      cout << "fit caught" << endl;
    }
}
```

[illegible] up [illegible] fit[illegible]

f( )[illegible] up [illegible] fit[illegible] g( )[illegible] version1[illegible] f( )[illegible] g( ) [illegible] version1[illegible] g( )[illegible] version2[illegible] g( )[illegible] version2[illegible] f( )[illegible] g( )[illegible] f( )[illegible] f( )[illegible]

my_unexpected( )[illegible] unexpected( )[illegible] unexpected( )[illegible] exit( )[illegible] abort( )[illegible] unexpected( )[illegible] unexcepted[illegible] unexcepted[illegible]

[illegible] new_handler( )[illegible] unexpected( )[illegible] terminate( )[illegible] unexpected( )[illegible]

[illegible] for[illegible]

□□□□□□□□□□□□□□□□□□□□□ for, while, do □ if □□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□ f( ) □□□□□□□□ up□ fit□□□□□□□□□□□□□□□□□□□□ f( ) □□□□□□□□□□□□□□□□□□□ g( )□ version2□□□□□ my_unexpected( ) □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□ set_unexpected( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□ unexcepted( )□□□□□□

17.3.4 □□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
void f ( ) □
```

□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
void  f ( )  throw (. . .)□  //  not  in  C++
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□□□□□□□ □□□ □□□□□□□□□□□□□□□□□□□□□□□

```
void  f ( )□
```

□□□ □□

17.3.5 □□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□ (á la C)□□□□□□□□□

```
catch (. . . ) {
    cout <<  "an exception was thrown" <<endl;
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□

17.3.6 □□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□ throw□□□□□

```
catch (. . .)  {
    cout <<  "an exception was thrown "<<endl;
    throw;
}
```

□□□□ catch□□

17.3.7 □□□□□□□

□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□“□□□□”□“□□□□”□□□

1. terminate( )

□□□□□□□□□□□□□□ terminate( )□□□□□□□□□□ unexception( )□□□□□□□□□□□□□□□□□□□□□□□□ C□□□□□□□□□□□□□ abort( )□□□□ abort( )□□

□□□ main( )□□□ (catch(. . .))□□

2. set_terminate( )

□□□□□□□□□□ set_ terminate( )□□□□□□□□□□ terminate( )□ set_ terminate( ) □□□□□ terminate( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ terminate( )□□□□□□□□□□□□□□□ void□□□□□□□□□□□□□ terminate( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ terminate( )□□□□□□□□□□□□□□□□

□□□□ unexpected( )□□□□□ terminate( )□□□□□□□□

□□□□□□□□□□□ set_ terminate( )□□□□□ set_ terminate( )□□□□□□□□□□ terminate( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□ terminate( )□□□□□□□□□□□□□□□□□□□□□

```
//: TRMNATOR.CPP -- Use of set_terminate()
// Also shows uncaught exceptions
#include <except.h>
#include <iostream.h>
#include <stdlib.h>

void terminator() {
  cout << "I'll be back!" << endl;
  abort();
}
```

```
void (*old_terminate)()
  = set_terminate(terminator);

class botch {
public:
  class fruit {};
  void f() {
    cout << "botch::f()" << endl;
    throw fruit();
  }
  ~botch() { throw 'c'; }
};

main() {
  try{
    botch b;
    b.f();
  } catch(...) {
    cout << "inside catch(...)" << endl;
  }
}
```

old_terminate□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ old_terminate□□□□□□□□□□□ set_ terminate( )□□

□ botch□□□□□ f( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ terminate( )□□□□□□□□□□□□□□□□□ catch(...)□□□□□□□□□□□□□□□□□□□□□□□□□ terminate( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ botch□□□□□□□□□□□□□□□□□□□□□□□ terminate( )□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

17.4 □□

□□□ C++□□□

□□ unexpected( )□□□□□□□□□□□□□□□□□□□□

```
//: CLEANUP.CPP -- Exceptions clean up objects
#include <fstream.h>
#include <except.h>
#include <string.h>

ofstream out("cleanup.out");
```

```
class noisy {
  static int i;
  int objnum;
  enum { sz = 40 };
  char name[sz];
public:
  noisy(const char* nm="array elem") throw(int){
    objnum = i++;
    memset(name, 0, sz);
    strncpy(name, nm, sz - 1);
    out << "constructing noisy " << objnum
      << " name [" << name << "]" << endl;
    if(objnum == 5) throw int(5);
    // Not in exception specification:
    if(*nm == 'z') throw char('z');
  }
  ~noisy() {
    out << "destructing noisy " << objnum
      << " name [" << name << "]" << endl;
  }
  void* operator new[](size_t sz) {
    out << "noisy::new[]" << endl;
    return ::new char[sz];
  }
  void operator delete[](void* p) {
    out << "noisy::delete[]" << endl;
    ::delete []p;
  }
};

int noisy::i = 0;

void unexpected_rethrow() {
  out << "inside unexpected_rethrow()" << endl;
  throw; // Rethrow same exception
}
main() {
  set_unexpected(unexpected_rethrow);
  try {
    noisy n1("before array");
    // Throws exception:
    noisy* array = new noisy[7];
    noisy n2("after array");
  } catch(int i) {
    out << "caught " << i << endl;
  }
```

```
   out << "testing unexpected:" << endl;
   try {
     noisy n3("before unexpected");
     noisy n4("z");
     noisy n5("after unexpected");
   } catch(char c) {
     out << "caught " << c << endl;
   }
 }.
```

□ noisy□□□□□□□□□□□□□□□□□□□□□□□□ noisy□□□□□□□□□□ i □□□□□□□□□□□□□□□□ objnum□□□□□□□□□□□□□□□□ name□□ □□ □ C□□□□□□ strncpy( )□□□

□□□“ z”□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ unexpected( )□□□

□□ new □ delete□□□□□□□□□□□□□□□□□□□□□□□□

□□ unexpected_rethrow( )□□□□□□□□□□□□□□□□□□□□□□□□ main( )□ □□□□□□□□ unexpected( )□□□□□□□□□□□□□□□□□□□ noisy□□□□□□□□□□ □□□□□□□□□□□□□□□ n2□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
constructing noisy 0 name [before array]
noisy::new[]
constructing noisy 1 name [array elem]
constructing noisy 2 name [array elem]
constructing noisy 3 name [array elem]
constructing noisy 4 name [array elem]
constructing noisy 5 name [array elem]
destructing noisy 4 name [array elem]
destructing noisy 3 name [array elem]
destructing noisy 2 name [array elem]
destructing noisy 1 name [array elem]
noisy::delete[]
destructing noisy 0 name [before array]
caught 5
testing unexpected:
constructing noisy 6 name [before unexpected]
constructing noisy 7 name [z]
inside unexpected_rethrow()
destructing noisy 6 name [before unexpected]
```

```
caught z
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□ 1~4□ □□□□□□□□

□□□□ new□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□ delete□ □□□□□□□□□□□□□□□ delete□ □□□□□□□□□□□□ □□□□□ new□ □□□□□□□□□□□□□□□□□□ 12□ □□□□□□□□□□□□□ □□□□ delete□ □□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□ n1□ □□□□□□□ n2□ □□□□□□□□□□□□□□□□□□

□□□□□ unexpected_rethrow( )□ □□□□□□□ n3□ □□□□□□ n4□ □□□□□□ □□□ □□□□□□□□□ unexpection( )□ □□□□□□□□□□□ unexpected_rethrow( )□□ □□□□ □ unexpected_rethrow( )□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□ n4□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□ n3□ □□□□□□□ □□□□□ unexpected_rethrow( )□ □□□□□□□□□□□□□ □□□ □□□□□□□□□□□□□□□□

17.5 □□□□

□□□□□□□□□□□□□□□□□□“ □□□□□□□□□□□□□□□□□□ ”□ □□□ □□ □□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
//: NUDEP.CPP -- Naked pointers
#include <fstream.h>
#include <stdlib.h>
ofstream out("nudep.out");

class bonk {
public:
  bonk() { out << "bonk()" << endl; }
  ~bonk() { out << "~bonk()" << endl; }
};

class og {
public:
  void* operator new(size_t sz) {
    out << "allocating an og" << endl;
    throw int(47);
    return 0;
  }
  void operator delete(void* p) {
    out << "deallocating an og" << endl;
    ::delete p;
  }
};

class useResources {
  bonk* bp;
  og* op;
public:
  useResources(int count = 1) {
    out << "useResources()" << endl;
    bp = new bonk[count];
    op = new og;
  }
  ~useResources() {
    out << "~useResources()" << endl;
    delete []bp; // Array delete
    delete op;
  }
};

main() {
  try {
    useResources ur(3);
  } catch(int) {
    out << "inside handler" << endl;
  }
};
```

□□□□

```
useResources()
bonk()
bonk()
bonk()
allocating an og
inside handler
```

□□□□useResources□□□□□□□□□□bonk□□□□□□□□□□□□□□□□□□□□□□□□□og::operator new□□useResources□□□□□□□□□□□□□□□□□□□□□□□useResources□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□bonk□□□□□□□□□□

□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“□□□□”□□□

[illegible]

```
//: WRAPPED.CPP -- Safe, atomic pointers
#include <fstream.h>
#include <stdlib.h>
ofstream out("wrapped.out");

// Simplified. Yours may have other arguments.
template<class T, int sz = 1> class pwrap {
  T* ptr;
public:
  class rangeError {}; // Exception class
  pwrap() {
    ptr = new T[sz];
    out << "pwrap constructor" << endl;
  }
  ~pwrap() {
    delete []ptr;
    out << "pwrap destructor" << endl;
  }
  T& operator[](int i) throw(rangeError) {
    if(i >= 0 && i < sz) return ptr[i];
    throw rangeError();
  }
};

class bonk {
public:
  bonk() { out << "bonk()" << endl; }
  ~bonk() { out << "~bonk()" << endl; }
  void g() {}
};

class og {
public:
  void* operator new[](size_t sz) {
    out << "allocating an og" << endl;
    throw int(47);
    return 0;
  }
  void operator delete[](void* p) {
    out << "deallocating an og" << endl;
    ::delete p;
  }
};

class useResources {
    pwrap<bonk, 3> Bonk;
    pwrap<og> Og;
  public:
    useResources() : Bonk(), Og() {
      out << "useResources()" << endl;
    }
    ~useResources() {
      out << "~useResources()" << endl;
    }
    void f() { Bonk[1].g(); }
  };

  main() {
    try {
      useResources ur;
    } catch(int) {
      out << "inside handler" << endl;
    } catch(...) {
      out << "inside catch(...)" << endl;
    }
  }
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ useResources□□

□□pwrap□□□□□□□□□□□□□□□□□□□ operator[]□□□□□□□□□□□□□□□□rangeError□□operator[]□□□□operator[]□□□□□□□□□□□0□□□□□0□□rangeError□□□

□□□□□□

```
bonk()
bonk()
bonk()
pwrap constructor
allocating an og
~bonk()
~bonk()
~bonk()
pwrap destructor
inside handler
```

□og□□□□□□□□□□□□□□□□□ bonk□□□□□□□□□□□□□□□□□□□□

17.6 □□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“□□”□□□□□□□□□□□□□□□□□□□□□□

□□“□□”□□□

```
//: AUTOEXCP.CPP -- No matching conversions
#include <iostream.h>
class except1 {};
class except2 {
public:
  except2(except1&) {}
};

void f() { throw except1(); }

main() {
  try { f();
  } catch (except2) {
    cout << "inside catch(except2)" << endl;
  } catch (except1) {
    cout << "inside catch(except1)" << endl;
  }
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□ except1□ □□□□ except2 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□ except1□ □□□

□□□□□□□□□□□□□□□□□□□□□□□

```
//: BASEXCPT.CPP -- Exception hierarchies
#include <iostream.h>

class X {
public:
  class trouble {};
  class small : public trouble {};
  class big : public trouble {};
  void f() { throw big(); }
};

main() {
  X x;
  try {
    x.f();
  } catch(X::trouble) {
    cout << "caught trouble" << endl;
  // Hidden by previous handler:
  } catch(X::small) {
    cout << "caught small trouble" << endl;
  } catch(X::big) {
    cout << "caught big trouble" << endl;
  }
}
```

□□□□□□□□□□□□□□□□□□□□□□□□ trouble□□□□ trouble□□□

□□□□□ small□ big□□□□ trouble□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“□□”□□

17.7 □□□□

□□ C++□□□

exception	□□□□□ C++□□□□□□□□□□□□□□ what()□□□□□□□□□□□
logic_erro	□□ exception□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
runtime_error	□□ exception□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□

I/O□□□□□ ios::failure□□ exception□□□□□□□□□□□□□□□□□

□□□

□ logic_error□□□□□	
domain_error	□□□□□□□□□
invalid_argument	□□□□□□□□□□□
length_error	□□□□□□□□□ NPOS□□□□□□□□□ NPOS□□□ size_t □□□□□□□□
out_of_range	□□□□□□
bad_cast	□□□□□□□□□□□□□□□ dynamic_cast□□□□□□ 18□□
bad_typeid	□□□□□□ typeid(*p)□□□□□□□ P□□□□□□□□□□□□□ 18□□

□ runtime_error□□□□□	
range_error	□□□□□□□□□
overflow_error	□□□□□□□□
bad_alloc	□□□□□□□□□□

17.8 [illegible]

[illegible] C [illegible]

17.8.1 [illegible]

[illegible]

1. [illegible]

[illegible] C [illegible] signal() [illegible]

[illegible]

2. [illegible]

[illegible] C++ [illegible] “ [illegible] ” [illegible] “ [illegible] ” [illegible] C++ [illegible]

3. [illegible]

[illegible] switch [illegible]

4. [illegible]

[illegible] assert() [illegible] abort() [illegible]

5. [illegible]

[illegible] try [illegible] catch(...) [illegible]

[illegible] try [illegible]

[illegible]

17.8.2 [illegible]

[illegible]

1) [illegible]

2) [illegible]

3) [illegible]

4) [illegible]

5) [illegible]

6) [illegible]

7) [illegible] C[illegible]

8) [illegible]

9) [illegible]

1. [illegible]

[illegible]

[illegible] unexpect() [illegible] unexpected() [illegible]

2. [illegible]

[illegible] C++ [illegible]

[illegible] expection [illegible] expection [illegible] what() [illegible]

3. [illegible]

[illegible] —— [illegible]

4. [illegible]

[illegible] —— [illegible]

[illegible] C++ [illegible]

5. [illegible]

[illegible] 15 [illegible]

6. [illegible] “[illegible]” [illegible] “[illegible]” [illegible]

[illegible] “[illegible]” [illegible]

[illegible] "[illegible]" [illegible]

```
//: CATCHREF.CPP -- Why catch by reference?
#include <iostream.h>

class base {
public:
  virtual void what() {
    cout << "base" << endl;
  }
};

class derived : public base {
public:
  void what() {
    cout << "derived" << endl;
  }
};

void f() { throw derived(); }

main() {
  try {
    f();
  } catch(base b) {
    b.what();
  }
  try {
    f();
  } catch(base& b) {
    b.what();
  }
}
```

[illegible]

```
base
derived
```

[illegible] ,[illegible] base[illegible] ([illegible])[illegible] base[illegible]

[illegible] ——[illegible]

7. □□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

1) □□□□□□□□□□□□□□□□□□□

2) □□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□ C□□□ C□□□□□□□□□□ C++□□

8. □□□□□□□□□□□□

□□ terminate()□□□□

□□ try□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

9. □□□□□□□□

□□□ 17.5.1□□□ WRAPPED.CPP□□

17.9 □□

□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□“□□□□”□□

17.10 □□

□□□□□□□□□□□□□□□□□□□□□□□□ C++□□

C++□□□

□□

17.11 □□

1) □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

[illegible] char*[illegible] try[illegible]

2) [illegible] 12[illegible] stash[illegible] operator[][illegible] out-of-range[illegible]

3) [illegible] main()[illegible]

4) [illegible] new[illegible] 10[illegible] 11[illegible] “[illegible]” [illegible] try[illegible] catch[illegible] while[illegible]

5) [illegible] terminate()[illegible]

6) [illegible]

7) [illegible]

8) [illegible]

□ 18□ □□□□□□□

□□□□□□□□ Run-time type identification, RTTI□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□

□□□□□□ C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□ □□□ □□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□ C++□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□ RTTI□□□□□□□□□□□□□□□□□□□□□□□□ C++□□□□□□□ □□□□□ RTTI□□□□□

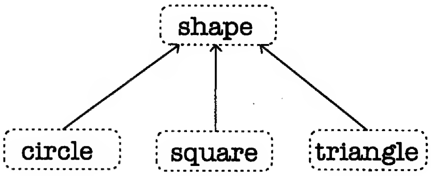
18.1 □□ ——shape

□□□□□□□□□□□□□□□□□□□□□ shape□□□□□□□□□ circle□ square □ triangle□

□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ shape □□□□ rhomboid□□□□□□□□□□□□□□□□□ shape□□□□□□□□□ draw()□ □□□□□□□□□□□□□□□ shape□□□□□ draw()□ draw()□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□ shape()□□□□□□□□□□□□□□□ □□□

□□□□□□□□□□□□ (circle□ square□ triangle)□□□□□□□□□□□ shape* □□□ □□□□□□□□□□□□□□□□□□□□□

18.2 □□□ RTTI


shape
circle
square
triangle

□ 18-1

□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□ □□□□ TurnColorIfYouAreA()□□□□□□□□□□□□□□□□□ shape::circle□ shape::square□ shape::triangle□□□

□□ □□□□□□□□□□□□□□ isA()□ type Of() □□□□□□□□□□□□□□□□□□ RTTI□□□□□□□□□□□□□□□□□□□□□□□□□“□□□□□□□ triangle□□□

□□□□□□□□”

□C++□□□RTTI□□□□□□□□C++□□□□

RTTI□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□RTTI□□□□□□□□□□

RTTI□□□□□□□

□□RTTI□□□□□□□□□□□□□sizeof()□□□□□□□□□□□□□□□□□□□□□□□□□□□typeid()□□□□□□□□□□□□□□□□□□□□□□□□typeinfo□□□□□□□□□□□□□□□□□□□“==”□“□=”□□□□□□□□□ □□□□□name()□□□□□□□□□□□□□□□typeid()□□□□shape*□□□□□□□□□□□shape*□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□s□□shape*,

```
cout << typeid(*s).name()<<endl;
```

□□□□s□□□□□□□□□

□□□□before(typeinfo&)□□□□typeinfo□□□□□□□□typeinfo□□□□□□□□□□□□□□□□，□□□□□true□false□□□□□

```
if(typeid(me).before(typeid(you)))  //...
```

□□□□□□□□□□□me□□□□□□□□□you□□□

RTTI□□□□□□“□□□□□□□□”□□□□“□□□□□”□□□□□□□□□□□□□□□□□□□□□□□□circle*□shape*□□□□□□□□□□□□shape*□□□□□circle*□□□□□□□□□□□□circle*□□□□shape*□□□□□□□□□□□□□□□□□shape*□□□□□circle*□□C□□□□□□□C++□□□□□static_cast,□□□□□□□□□□□□□□□□□□□□“□□□□□□□□□□circle*□□□□□□□□□□□”□□□□□□□□□□□□□□□circle□□□□□□□□□□□□□□□□□□□RTTI□□□□□□□□□□□□□□□□□□shape*□□□□□circle*(□□□□)□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□null□□□□□□□□□□circle*□□□

C++□RTTI□“□□□□□□□□□”□□□□□□“□□□□”□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□dynamic_cast□□□□□□□□□

```
shape* sp = new circle;
circle* cp = dynamic_cast<circle*>(sp);
if(cp) cout << “cast successful”;
```

□□□

□□□□□□□□□□□□□□□□□□□□□□□□□，□□□□□□□□□shape□□□□□□□□□□□□□□□

```
circle* cp = dynamic_cast<circle*>(sh);
square* sp = dynamic_cast<square*>(sh);
triangle* tp = dynamic_cast<triangle*>(sh);
```

□□□□□□□——□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□ shape□ □

```
//: RTSHAPES.CPP -- Counting shapes
#include <iostream.h>
#include <time.h>
#include <typeinfo.h>
#include "..\14\tstash.h"

class shape {
protected:
  static int count;
public:
  shape() { count++; }
  virtual ~shape() = 0 { count--; }
  virtual void draw() const = 0;
  static int quantity() { return count; }
};

int shape::count = 0;

class rectangle : public shape {
  void operator=(rectangle&); // Disallow
protected:
  static int count;
public:
  rectangle() { count++; }
  rectangle(const rectangle&) { count++;}
  ~rectangle() { count--; }
  void draw() const {
    cout << "rectangle::draw()" << endl;
  }
  static int quantity() { return count; }
};

int rectangle::count = 0;

class ellipse : public shape {
  void operator=(ellipse&); // Disallow
protected:
  static int count;
public:
  ellipse() { count++; }
  ellipse(const ellipse&) { count++; }
  ~ellipse() { count--; }
  void draw() const {
    cout << "ellipse::draw()" << endl;
  }
  static int quantity() { return count; }
};

int ellipse::count = 0;

class circle : public ellipse {
  void operator=(circle&); // Disallow
protected:
  static int count;
public:
  circle() { count++; }
  circle(const circle&) { count++; }
  ~circle() { count--; }
  void draw() const {
    cout << "circle::draw()" << endl;
  }
  static int quantity() { return count; }
};

int circle::count = 0;

main() {
  tstash<shape> shapes;
  time_t t;
  // Seed random number generator:
  srand((unsigned)time(&t));
  const mod = 12;
  for(int i = 0; i < rand() % mod; i++)
    shapes.add(new rectangle);
  for(int j = 0; j < rand() % mod; j++)
    shapes.add(new ellipse);
  for(int k = 0; k < rand() % mod; k++)
    shapes.add(new circle);
  int Ncircles = 0;
  int Nellipses = 0;
  int Nrects = 0;
  int Nshapes = 0;
  for(int u = 0; u < shapes.count(); u++) {
    shapes[u]->draw();
    if(dynamic_cast<circle*>(shapes[u]))
      Ncircles++;
    if(dynamic_cast<ellipse*>(shapes[u]))
      Nellipses++;
    if(dynamic_cast<rectangle*>(shapes[u]))
      Nrects++;
      if(dynamic_cast<shape*>(shapes[u]))
        Nshapes++;
    }
    cout << endl << endl
         << "circles = " << Ncircles << endl
         << "ellipses = " << Nellipses << endl
         << "rectangles = " << Nrects << endl
         << "shapes = " << Nshapes << endl
         << endl
         << "circle::quantity() = "
         << circle::quantity() << endl
         << "ellipse::quantity() = "
         << ellipse::quantity() << endl
         << "rectangle::quantity() = "
         << rectangle::quantity() << endl
         << "shape::quantity() = "
         << shape::quantity() << endl;
  }
```

□□□ RTTI□□□□□□□□□□□□□

18.3 □□□□

□□□□□□ RTTI□□□□□□□□□□□□□□□□□□□□□

18.3.1 □□□□□□□ typeid()

□□□□□□□□ typeid()□□□□□□□□□□□□□□□□□□□□□□ true□

```
typeid(47) == typeid(int)
typeid(0) == typeid(int)
int i;
typeid(i) == typeid(int)
typeid(&i) == typeid(int*)
```

18.3.2 □□□□□□□□□

typeid()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
//: RNEST.CPP -- Nesting and RTTI
#include <iostream.h>
#include <typeinfo.h>

class one {
  class nested {};
  nested* n;
public:
  one() { n = new nested; }
  ~one() { delete n; }
  nested* N() { return n; }
};

main() {
  one O;
  cout << typeid(*O.N()).name() << endl;
}
```

typeinfo::name()□□□□□□□□□□□□□□□□□□□ one::nested□

18.3.3 □□□□□

□□typeid( )□□□

```
class X {
 int i;
public:
  // ...
};
class Y : public X {
  int j;
public:
  // ...
};
```

□□□□□□□□□□□□□□□□□□

```
X* xp = new Y;
```

typeid()□□□

```
typeid(*xp) == typeid(X)
typeid(*xp) != typeid(Y)
```

□□□□RTTI□□□□□□□

18.3.4 □□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
class d1 {
public:
  virtual void foo() {}
};
class d2 {
public:
  virtual void bar() {}
};
class mi : public d1, public d2 { };
class mi2 : public mi {};

d2* D2 = new mi2;
mi2* MI2 = dynamic_cast<mi2*>(D2);
mi* MI = dynamic_cast<mi*>(D2);
```

□□□□□□□□□□□□□□□□□□□□□□□□□ mi2□□ mi□ mi2□

□□□□□□□□□□□□□□□□□□

```
d1* D1 = dynamic_cast<d1*>(D2);
```

□□□□□□□□□□□ D2□□□□□□□ mi2□□□□□□□□□ d1□□□□□□□

□□□□□□□ dynamic_cast□ typeid()□□□□□□□□□□□□□ typeid()□□□□□□ typeinfo □□ true□□ typeid()□□□ dynamic_cast□□□ D2□□□□□□□□□□□□

```
typeid(D2) != typeid(mi2*)
```

D2□□□□□□□□□□□□

```
typeid(D2) == typeid(d2*)
```

18.3.5 void□ □

□□□□□□□□□□□ void□□□□□□□□

```
//: VOIDRTTI.CPP -- RTTI & void pointers
#include <iostream.h>
#include <typeinfo.h>

class stimpy {
public:
  virtual void happy() {}
  virtual void joy() {}
};

main() {
  void* v = new stimpy;
  // Error:
//!  stimpy* s = dynamic_cast<stimpy*>(v);
  // Error:
//!  cout << typeid(*v).name() << endl;
}
```

void*□□□□□ “ □□□□□□□□ ”□

18.3.6 □□□□□□ RTTI

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ RTTI□ □

□□□□□□□□□□□□□□□□□□□□□□□ 13□ □□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□

```
//: INHORDER.CPP -- Order of constructor calls
#include <iostream.h>
#include <typeinfo.h>

template<int id> class announce {
public:
  announce() {
    cout << typeid(*this).name()
         << " constructor " << endl;
  }
  ~announce() {
    cout << typeid(*this).name()
         << " destructor " << endl;
  }
};

class X : public announce<0> {
  announce<1> m1;
  announce<2> m2;
public:
  X() { cout << "X::X()" << endl; }
  ~X() { cout << "X::~X()" << endl; }
};

main() { X x; }
```

TYPEINFO.H□ □□□□□□□□□□□□□ typeid()□ □ □ typeinfo□ □□□□□□□□□
□□□
RTTI□ □□□□□□□□□□□□□□□ X□ □□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□

□□□□□□□□□□□ C++□ □□□□□□□□

18.4 □□

RTTI□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□

```
class B {
public:
  virtual float f() { return 1.0;}
};
class D : public B { /* ... */ };
B* p = new D;
B& r = *p;
```

typeid()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
typeid(p) == typeid(B*)
typeid(p) != typeid(D*)
typeid(r) == typeid(D)
```

□□□□□□□□□□□□□ typeid()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
typeid(*p) == typeid(D)
typeid(*p) != typeid(B)
typeid(&r) == typeid(B*)
typeid(&r) != typeid(D*)
```

□□□□□□□ typeid()□□□□□□□□□□□□□□□

```
typeid(r.f()) == typeid(float)
```

□□

□□□ C++□□□□□□ bad-cast□□□□□□□□□□□□□□□□□□□□□□□

```
class X {};

mi MI;
d1 & D1 = MI; // upcast to reference
try {
  X& xr = dynamic_cast<X&>(D1);
} catch(...) {
  cout << "dynamic_cast<X&>(D1) failed"
       << endl;
}
```

□□□□□□□□□□ D1□□□□□□□□ X□□□□□□□□□□□□□□□□ xr□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□ typeid()□□□□□□□□□□□□□□□□□□□□□□□□□ C++□□□□□□□ bad-typeid:

```
B* bp = 0;
try {
  typeid(*bp); // throws exception
} catch(bad_typeid) {
  cout << "Bad typeid() expression" << endl;
}
```

□□□□□ typeid□□

18.5 □□□□

□□□ RTTI□□□□□□□□□□□□□□□□□□ virtual□□□

```
//: MIRTTI.CPP -- MI & RTTI
#include <iostream.h>
#include <typeinfo.h>

class BB {
public:
  virtual void f() {}
};
class B1 : virtual public BB {};
class B2 : virtual public BB {};
class MI : public B1, public B2 {};

main() {
  BB* bbp = new MI; // Upcast
  // Proper name detection:
  cout << typeid(*bbp).name() << endl;
  // Dynamic_cast works properly:
  MI* mip = dynamic_cast<MI*>(bbp);
  // Can't force old-style cast:
  //! MI* mip2 = (MI*)bbp; // Compile error
}
```

□□□□□□□virtual□□□□□typeid()□□

```
MI* mip = (MI*)bbp; // compile-time error
```

□□□□□□□□□□□□□□□□□□□□□□□□□

18.6 □□□□RTTI

□□RTTI□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□switch□□□□□□□□□□□□□□□RTTI□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C++□□□□□□□□□□□□□□□□□□□□□RTTI□

□□RTTI□□switch□□□

□□14□□14.6□□□WINDS.CPP□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□instrument

 ClearSpitvalve() percussion electronic RTTI wind

 RTTI RTTI

##

 15 trash recycling RTTI

```
//: RECYCLE2.CPP -- Chapter 14 example w/ RTTI
#include <fstream.h>
#include <stdlib.h>
#include <time.h>
#include <typeinfo.h>
#include "..\14\tstack.h"
ofstream out("recycle2.out");

class trash {
  float Weight;
public:
  trash(float Wt) : Weight(Wt) {}
  virtual float value() const = 0;
  float weight() const { return Weight; }
  virtual ~trash() {}
};

class aluminum : public trash {
  static float val;
public:
  aluminum(float Wt) : trash(Wt) {}
  float value() const { return val; }
  static void value(int newval) {
    val = newval;
  }
};

float aluminum::val = 1.67;
class paper : public trash {
  static float val;
public:
  paper(float Wt) : trash(Wt) {}
  float value() const { return val; }
  static void value(int newval) {
      val = newval;
    }
  };

  float paper::val = 0.10;

  class glass : public trash {
    static float val;
  public:
    glass(float Wt) : trash(Wt) {}
    float value() const { return val; }
    static void value(int newval) {
      val = newval;
    }
  };

  float glass::val = 0.23;

  // Sums up the value of the trash in a bin:
  template<class T> void
  SumValue(const tstack<T>& bin, ostream& os) {
    tstackIterator<T> tally(bin);
    float val = 0;
    while(tally) {
      val += tally->weight() * tally->value();
      os << "weight of "
          << typeid(*tally.current()).name()
          << " = " << tally->weight() << endl;
      tally++;
    }
    os << "Total value = " << val << endl;
  }

  main() {
    // Seed the random number generator
    time_t t;
    srand((unsigned)time(&t));

    tstack<trash> bin; // Default to ownership
    // Fill up the trash bin:
    for(int i = 0; i < 30; i++)
      switch(rand() % 3) {
        case 0 :
          bin.push(new aluminum(rand() % 100));
          break;
        case 1 :
```

```
        bin.push(new paper(rand() % 100));
        break;
      case 2 :
        bin.push(new glass(rand() % 100));
        break;
    }
  // Note difference w/ chapter 14: Bins hold
  // exact type of object, not base type:
  tstack<glass> glassbin(0); // No ownership
  tstack<paper> paperbin(0);
  tstack<aluminum> ALbin(0);
  tstackIterator<trash> sorter(bin);
  // Sort the trash:
  while(sorter) {
    aluminum* ap =
      dynamic_cast<aluminum*>(sorter.current());
    paper* pp =
      dynamic_cast<paper*>(sorter.current());
    glass* gp =
      dynamic_cast<glass*>(sorter.current());
    if(ap) ALbin.push(ap);
    if(pp) paperbin.push(pp);
    if(gp) glassbin.push(gp);
    sorter++;
  }
  SumValue(ALbin, out);
  SumValue(paperbin, out);
  SumValue(glassbin, out);
  SumValue(bin, out);
}
```

□□RTTI□□□□□□15□□□□RTTI□□□□□□□□□□□□□□□□□□□□□□□□□□□C++□□□□RTTI□□□□

18.7 RTTI□□□□□□

□□□RTTI□□□□VTABLE□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□typeinfo□□□□□□□□□□□□typeinfo□□□□□□□typeid()□□□□□□□□□□□□□□VPTR□□□typeinfo□□□□□□□□□□□□typeinfo□□□□□□□——□□□□□□□□□□□——□□□□□□□□□□□□□□□□□

□□dynamic_cast<□□*><□□□>□□□□□□□□□□□□□□□□□□RTTI□□□□□□□*□□□RTTI□□□□□□□□□□□□□□□□□□□□□□□*□□□□□□□*□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ typeid()□□□

18.8 □□□□□□□RTTI

□□□□□□□□□RTTI□□□□□□□□□□□□□□□□RTTI□□□ RTTI□□□□ C++□□RTTI□□□□□□□□□□□C++□□□□□“□□”□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□RTTI□□□RTTI□□ dynacast()□□□□□□□□□□□□□□□□□□□□□□ dynamic_type()□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
//: SELFRTTI.CPP -- Your own RTTI system
#include "..\14\tstack.h"
#include <iostream.h>
class security {
protected:
  enum { baseID = 1000 };
public:
  virtual int dynamic_type(int ID) {
    if(ID == baseID) return 1;
    return 0;
  }
};

class stock : public security {
protected:
  enum { typeID = baseID + 1 };
public:
  int dynamic_type(int ID) {
    if(ID == typeID) return 1;
    return security::dynamic_type(ID);
  }
  static stock* dynacast(security* s) {
    if(s->dynamic_type(typeID))
      return (stock*)s;
    return 0;
  }
};

class bond : public security {
```

```
protected:
  enum { typeID = baseID + 2 };
public:
  int dynamic_type(int ID) {
    if(ID == typeID) return 1;
    return security::dynamic_type(ID);
  }
  static bond* dynacast(security* s) {
    if(s->dynamic_type(typeID))
      return (bond*)s;
    return 0;
  }
};

class commodity : public security {
protected:
  enum { typeID = baseID + 3};
public:
  int dynamic_type(int ID) {
    if(ID == typeID) return 1;
    return security::dynamic_type(ID);
  }
  static commodity* dynacast(security* s) {
    if(s->dynamic_type(typeID))
      return (commodity*)s;
    return 0;
  }
  void special() {
    cout << "special commodity function\n";
  }
};

class metal : public commodity {
protected:
  enum { typeID = baseID + 4};
public:
  int dynamic_type(int ID) {
    if(ID == typeID) return 1;
    return commodity::dynamic_type(ID);
  }
  static metal* dynacast(security* s) {
    if(s->dynamic_type(typeID))
      return (metal*)s;
    return 0;
  }
```

```
};

main() {
  tstack<security> portfolio;
  portfolio.push(new metal);
  portfolio.push(new commodity);
  portfolio.push(new bond);
  portfolio.push(new stock);
  tstackIterator<security> it(portfolio);
  while(it) {
    commodity* cm =
      commodity::dynacast(it.current());
    if(cm) cm->special();
    else cout << "not a commodity" << endl;
    it++;
  }
  cout << "cast from intermediate pointer:\n";
  security* sp = new metal;
  commodity* cp = commodity::dynacast(sp);
  if(cp) cout << "it's a commodity\n";
  metal* mp = metal::dynacast(sp);
  if(mp) cout << "it's a metal too!\n";
}
```

[illegible] typeID[illegible] dynamic_type()[illegible] typeID[illegible] dynacast()[illegible] ——[illegible]

[illegible] security[illegible] typeID[illegible] baseID[illegible] baseID[illegible] enum[illegible] protected[illegible]

enum baseID[illegible] security[illegible] ID[illegible] ID[illegible] ID[illegible] ID[illegible] ID[illegible] protected[illegible]

[illegible] RTTI[illegible] metal[illegible] commodity[illegible] commodity[illegible] special()[illegible] metal[illegible] special()[illegible] dynamic_type()[illegible] metal[illegible] commodity[illegible] “[illegible]”[illegible] RTTI[illegible]

dynacast()[illegible] dynamic_type()[illegible] typeID[illegible] dynamic_type()[illegible] typeID[illegible]

[illegible] dynamic_type()[illegible] dynamic_type()[illegible] ID[illegible] dynamic_type()[illegible]

[illegible] dynamic_type()[illegible] 1(true)[illegible] dynacast()[illegible] security[illegible] false,[illegible] dynacast()[illegible] C++[illegible] dynamic_cast[illegible]

C++[illegible]

[illegible] dynamic_cast [illegible] DYNAMIC-CAST[illegible]

18.9 [illegible]

[illegible][1][illegible]

[illegible] C[illegible] C[illegible] void[illegible]

[illegible] C++[illegible] dynamic_cast([illegible])[illegible] const_cast[illegible] static_cast[illegible] reinterpret_cast[illegible]

[illegible] newtype[illegible]

[illegible]

static_cast	[illegible]“[illegible]”[illegible]“[illegible]”[illegible]
const_cast	[illegible] const[illegible] volatile[illegible]
dynamic_cast	[illegible]
reinterpret_cast	[illegible]

[illegible]

---

[1] [illegible] Josée Lajoie“The new cast notation and the bool data type” [illegible] C++[illegible] 1994[illegible] 9[illegible] pp.46-51.

□□

18.9.1 static_cast

static_cast□□□□□□□□□□□□□□□□“□□”□□□□□□□□□“□□”□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□static_cast□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□void*□□□□□□□□□□□□□□□□□□□□□□

```
//: STATCAST.CPP -- Examples of static_cast

class base { /* ... */ };
class derived : public base {
public:
  // ...
  // Automatic type conversion:
  operator int() { return 1; }
};

void func(int) {}

class other {};

main() {
  int i = 0x7fff; // Max pos value = 32767
  long l;
  float f;
  //(1) typical castless conversions:
  l = i;
  f = i;
  // Also works:
  l = static_cast<long>(i);
  f = static_cast<float>(i);

  //(2) narrowing conversions:
  i = l; // May lose digits
  i = f; // May lose info
 // Says "I know," eliminates warnings:
 i = static_cast<int>(l);
 i = static_cast<int>(f);
 char c = static_cast<char>(i);

 //(3) forcing a conversion from void* :
 void* vp = &i;
 // Old way produces a dangerous conversion:
 float* fp = (float*)vp;
 // The new way is equally dangerous:
 fp = static_cast<float*>(vp);
```

```
    //(4) implicit type conversions, normally
    // Performed by the compiler:
    derived d;
    base* bp = &d; // Upcast: normal and OK
    bp = static_cast<base*>(&d); // More explicit
    int x = d; // Automatic type conversion
    x'= static_cast<int>(d); // More explicit
    func(d); // Automatic type conversion
    func(static_cast<int>(d)); // More explicit

    //(5) Static Navigation of class hierarchies:
    derived* dp = static_cast<derived*>(bp);
    // ONLY an efficiency hack. dynamic_cast is
    // Always safer. However:
    // Other* op = static_cast<other*>(bp);
    // Conveniently gives an error message, while
    other* op2 = (other*)bp;
    // Does not.
  }
```

□□□1□□□□□□□□□□C□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ int□□□ long□ float□□□□□□□□□□□□□□□ int□□□□□□□□□□□□□ static_cast□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□2□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ int□□□□ long□ float□□“□”□□□□□□□□□“□□□□”□□

□ C++□□□ void*□□□□□□□□□□□□□□ C□□□□□□□□□□ 3□□ static_cast□□□□□□□□□□□□□□□

□□4□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ static_cast□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□ dynamic_cast□□□□□□□□□□□□□ 5□□□□□□□□□□□□□□ static_cast□□□ dynamic_cast□□□□□□□□

18.9.2 const_cast

□□□□□□ const □□□□ const□□□□□ volatile□□□□□□ volatile□□□□□ const_cast□□□□□□ const_cast□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
  //: CONSTCST.CPP -- Const casts

  main() {
```

```
  const int i = 0;
  int* j = (int*)&i; // Deprecated form
  j  = const_cast<int*>(&i); // Preferred
  // Can't do simultaneous additional casting:
//! long* l = const_cast<long*>(&i); // Error
  volatile int k = 0;
  int* u = const_cast<int*>(&k);
}

class X {
  int i;
// mutable int i; // a better approach
public:
  void f() const {
    // Casting away const-ness:
    (const_cast<X*>(this))->i = 1;
  }
};
```

□□□□□□ const□ □□□□□□□□□□□□□□ const□ □□□□□□□□□□□□□□□□□ const□ □□□□□□□□□□□□□□□□□□□□□ const_cast□ □□□□□□□□□ volatile□

□□□□□□ const□ □□□□□□□□□□□□□□□□□□□□□ X*□ this□ □□□□□□□□□□□□□□□□□□ const_cast□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□ mutable□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ const□ □□□□□□□

18.9.3 reinterpret_cast

□□□ const_cast□ reinterpret_cast□ □□□□□□□□□□□□□□□□□

reinterpret_cast□ □□ C□ □□□□□□□□□□□□□□□□□□□□□□□□□□□ reinterpret_cast□ □□□□□□□□□□

```
//: REINTERP.CPP -- Reinterpret_cast
// Example depends on VPTR location,
// Which may differ between compilers.
#include <string.h>
#include <fstream.h>
ofstream out("reinterp.out");

class X {
  enum { sz = 5 };
  int a[sz];
public:
  X() { memset(a, 0, sz * sizeof(int)); }
  virtual void f() {}
  // Size of all the data members:
  int membsize() { return sizeof(a); }
  friend ostream&
    operator<<(ostream& os, const X& x) {
      for(int i = 0; i < sz; i++)
        os << x.a[i] << ' ';
      return os;
  }
};

main() {
  X x;
  out << x << endl; // Initialized to zeroes
  int* xp = reinterpret_cast<int*>(&x);
  xp[1] = 47;
  out << x << endl; // Oops!

  X x2;
 const vptr_size = sizeof(X) - x2.membsize();
 long l = reinterpret_cast<long>(&x2);
 // *IF* the VPTR is first in the object:
 l += vptr_size; // Move past VPTR
 xp = reinterpret_cast<int*>(l);
 xp[1] = 47;
 out << x2 << endl;
}
```

□X□□□□□□□□□□□□□main()□□□□X□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□reinterpret_cast□□□□□int*□□□□□□□int*□□□□□□□□□□□□□□□□□□□□□□1□□□47□□□□□□□□□□□□□□[1]□□□

```
00  0  0  0
47  0  0  0  0
```

□□VPTR□□□□□□□□□□□□□□xp[0]□□□□xp[1],□□□VPTR□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□VPTR□□□□□□□□□□□□□□□□□□□□long□(□reinterpret_cast)□□□VPTR□□int*□□□□□□□□□□□□□□□□□

```
0  47  0  0  0
```

---

[1] □□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□ reinterpret_cast□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□

18.10 □□

RTTI□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ RTTI□□□□ □□□□□ RTTI□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□

1) □□□□□□□□□□□□□□□□

2) □□□□□□□□□□□□□□□□□

3) □□□□□□□□□□□□□□□□□□□□□□□ RTTI□□□

4) □□□□□□□□□□□□□□□□□□□□□□□□

□□ RTTI□□□□□□□ C++□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ RTTI□□□□□ □□□□ C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□ □□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□

18.11 □□

1. □ RTTI□□□□□□□□□□□□□□□ typeid()□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□

2. □ RTTI□□□□□□□□ TurnColorIfYouAreA()□□□

3. □□ 14□□ WIND5.CPP□□□□□□□□□□□□□□□□□ instrument□□□□□ wind□□□□□□□□ ClearSpitValve()□□□□□ wind□□□□□□□□□□□□ tstash□□ □□□□□□□ instrument□□□□□ new□□□□□ instrument□□□□□□□□□□ RTTI□ □□□□□□□□□□□□□□□□□ wind□□□□□□□□□□□□□□□□□□ ClearSpitValve()□□□□□□□□ instrument□□□□□□□ ClearSpitValve()□□□□□□□ □□□□□

[illegible] A [illegible]

[illegible] C++[illegible]

A.1 bool[illegible] true[illegible] false

[illegible] [1][illegible] typedef[illegible] typedef[illegible] typedef[illegible] int[illegible] int[illegible]

[illegible] bool[illegible]

[illegible] bool[illegible] true[illegible] 1[illegible] false([illegible] 0)[illegible]

[illegible]	bool[illegible]
&&‖ [illegible]	[illegible] bool[illegible] bool[illegible]
< > <= >= == !=	[illegible] bool[illegible]
if ,for while,do	[illegible] bool[illegible]
?:	[illegible] bool[illegible]

[illegible] int[illegible] int[illegible] bool[illegible]

[illegible]“[illegible]”[illegible] ++[illegible] true[illegible] enum[illegible] bool [illegible] int[illegible] bool[illegible] 0~1[illegible]

[illegible] bool[illegible]

A.2 [illegible]

[illegible] C++[illegible] C++[illegible] IOSTREAM.H[illegible]

```
# include  <iostream>
```

---

[1] [illegible]“ Josée Lajoie[illegible] "The new cast notation and the bool data type”C++[illegible] 1994[illegible] 9[illegible]

 C .h
 C++ C

A.3   C++

 C++ C
 C C C++

  Internet BBS

 6 C++

 <climits> <cfloat> <new>
 bad_alloc set_new_handler
 RTTI <typeinfo> terminate() unexpected() <exception>

 C++ <stdexcept>
 <cassert> C ASSERT.H

 C++
 != > <= >= tuple pair
 STL STL

 C
 append() assign()
insert() remove() replace() resize() copy() find() rfind find_first_of()
find_last_of() find_first_not_of() find_last_not_of() substr() compare()
 = += [] “ ” wstring
string wstring (<string> C STRING.H) basic_string

 strstream 6

 <bits> <bitstring>
bits bit_string bits bit_string 5 bits
 set() reset()
count() length() test() any() none() to_ushort()
to_ulong() to_string()

bit_string bits
 string bits bit_string bits
 0 bit_string bits
bit_string bits bit_string

[illegible] STL[illegible]

[illegible] STL[illegible] adjacent_find□ prev_permutation□ binary_search□ push_heap□ copy□ random_shuffle□ copy_backward□ remove□ count□ remove_copy□ count_if□ remove_copy_if□ equal,□ remove_if□ equal_range□ replace□ fill□ replace_copy□ fill_n□ replace_copy_if□ find□ replace_if□ find_if□ reverse□ for_each□ reverse_copy□ generate□ rotate□ generate_n□ rotate_copy□ includes□ search□ inplace_merge□ set_difference□ lexicographical_compare□ set_intersection□ lower_bound□ set_symmetric_difference□ make_heap□ set_union□ max□ sort□ max_element□ sort_heap□ merge□ stable_partition□ min□ stable_sort□ min_element□ swap□ mismatch□ swap_ranges□ next_permutation□ transform□ nth_element□ unique□ partial_sort□ unique_copy□ partial_sort_copy□ upper_bound□ partition□

[illegible] C++[illegible] float□ double□ long double[illegible]

A.4 [illegible] STL[illegible]

[illegible] 15[illegible] C++[illegible] C++□ STL [1] [illegible] STL[illegible]

[illegible] STL[illegible] STL[illegible]

[illegible] STL[illegible] STL[illegible] STL[illegible]

[illegible] STL□ set[illegible] set[illegible] int[illegible]

```
//: INTSET.CPP -- Simple use of STL set
#include <set.h>

void main() {
  set<int, less<int> > intset;
  for(int i = 0; i < 25; i++)
    for(int j = 0; j < 10; j++)
      // Try to insert multiple copies:
      intset.insert(j);
  // Print to output:
  copy(intset.begin(), intset.end(),
```

---

[1] □ Alexander Stepanov□ Meng Lee[illegible] C++[illegible]

```
        ostream_iterator<int>(cout, "\n"));
}
```

set□□insert()□□□□□□□□□□□□□□□□□□□□□□□□□

copy()□□□□□□□□□□□□set□□□□□begin()□end()□□□□□□□□□□□□□□□copy()□□□int□□□□□cout□□□□□□□□□□□□

copy()□□copy()□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□INTSET.CPP□□□□□□□□□□□□15□□□□□□□□□□□□□□□□□□□□□

```
//: CONCORD.CPP -- Concordance with STL
#include <set.h>
#include <fstream.h>
#include "..\allege.h"
#include "..\14\sstring.h"
const char* delimiters =
  " \t;()\"<>:{}[]+-=&*#.,/\\"
  "0123456789";

main(int argc, char* argv[]) {
  allege(argc == 2, "usage: concord filename");
  ifstream in(argv[1]);
  allegefile(in);
  set<Hstring, less<Hstring> > concordance;
  const sz = 1024;
  char buf[sz];
  while(in.getline(buf,sz)) {
    // Capture individual words:
    char* s = strtok(buf, delimiters);
    while(s) {
      concordance.insert(s); // Auto type conv.
      s = strtok(0, delimiters);
    }
  }
  // output results:
  copy(concordance.begin(), concordance.end(),
       ostream_iterator<Hstring>(cout, "\n"));
}
```

□□□□□□□□□□□□□Hstring□□□int□□□□□□□□□□□□□□□□□strtok()□□□□□□□□□□INTSET.CPP□□□□□□□□□□□□□□□□□□□□

[illegible]

STL[illegible]

[illegible]	[illegible]
vector	[illegible] /[illegible] /[illegible] bool[illegible]
list	[illegible] /[illegible]
deque	[illegible] /[illegible] /[illegible]
stack	[illegible] vector[illegible] list[illegible] deque[illegible]
queue	[illegible] list[illegible] deque[illegible]
priority_queue	[illegible] top(),push()[illegible] pop()[illegible] vector[illegible] deque[illegible]
multiset	[illegible]
map	[illegible]
multimap	[illegible] (hash)[illegible]

[illegible] STL[illegible] STL[illegible]

A.5 asm[illegible]

[illegible] C++[illegible] C++[illegible] C++[illegible]

A.6 [illegible]

[illegible] &[illegible] |[illegible] ……[illegible] C[illegible] C++[illegible]

[illegible]	[illegible]
and	&&([illegible])
or	‖[illegible]
not	![illegible]
not_eq	!=[illegible]
bitand	&[illegible]
and_eq	&=[illegible]
bit or	|[illegible]
or_eq	|=[illegible]
xor	^[illegible]
xor_eq	^=[illegible]
compl	~[illegible] 1[illegible]

□□B □□□□

□□□□[1]□□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□ Dan Saks□□Tom Plum□□□“ C++ Programming Guidelines”□Plum Hall 1991□□Scott Meyers(“ Effective C++”□□□□□□ Addison-Wesley,1992)□Rob Murray□“ C++ strategies & Tactics”□□□□ Addision-Wesley,1993□□□□□□□□□□□□□□□□□□□□

1. □□□C++□□□□□□□□□C□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C□□□C++□□□□□□□□□□□□□ C++□□□□□□□□□□□□□□□□□□

2. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“ □□ ”□□

3. □□

4. □□ private, □□□□□□□□public□□□□□□□□□□□□□□□□□□□□□□□□□□□ public□□□□□□□□□□□□□□□□□□□□□□□□□□□□ private□□□□□□□□□□□□□□□□□□□□□protected□□

5. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ C++□□

6. □□

7. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□(Andrew Koenig□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

8. □□□

9. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

---

[1] □□□□□□□ Andrew Binstock□□□□□□ Unix Review□□□□□□□□□□

[illegible]

10. [illegible]——[illegible]

11. [illegible]

12. [illegible]

13. [illegible]

14. [illegible]“ is-a”[illegible]“ has-a”[illegible]

15. [illegible]

16. [illegible] protected[illegible]

17. [illegible]

18. [illegible] virtual[illegible]

19. [illegible]

20. [illegible]

21. [illegible]

22. [illegible] C++[illegible]

23. [illegible]“ [illegible] ”[illegible]

[illegible] 11[illegible]

24. [illegible]

25. [illegible] “ =” [illegible] “ =” [illegible]

26. [illegible] “ =” [illegible]

27. [illegible] 3[illegible] /Cheshire cat[illegible]

28. [illegible]

29. [illegible]

30. [illegible]

31. [illegible] ostream[illegible] ostream[illegible]

```
class ostream
```

[illegible] ostream[illegible] typedef[illegible]

```
#include <iostream.h>
```

[illegible]

32. [illegible] “ =” [illegible] x=x[illegible] *this[illegible] A=B=C[illegible]

33. [illegible] const[illegible]

34. [illegible] return[illegible]

```
return foo(i,j);
```

[illegible]“[illegible]”[illegible]

```
foo  x(i,j);
return  x;
```

[illegible]

35. [illegible]

36. [illegible]

37. [illegible]

38. [illegible]C++[illegible]

39. [illegible]

40. [illegible]

41. [illegible]

42. [illegible]STDIO.H[illegible]printf()[illegible]6[illegible]C++[illegible]C[illegible]

43. [illegible]C[illegible]C++[illegible]C++[illegible]

44. [illegible]‘ *’[illegible]

45. [illegible]“ =”[illegible]

46. [illegible]“ foo a=b;”[illegible]

[illegible]“ =”[illegible]“ foo a(b);”[illegible]

47. [illegible] C++[illegible] C[illegible]

48. [illegible] C++[illegible]

49. [illegible]

50. [illegible] assert()[illegible]

51. [illegible]

52. [illegible] set_unexpected() [illegible] unexpected()[illegible] unexpected()[illegible]

53. [illegible] terminate()[illegible]

54. [illegible] terminate()[illegible]

55. [illegible]“ [illegible] ”[illegible]

56. [illegible] for[illegible] for[illegible] for[illegible] for[illegible]

57. [illegible]

58. [illegible] new[illegible] 0[illegible] 0[illegible]

59. [illegible] new[illegible] delete[illegible] new[illegible] delete[illegible] new[illegible] delete[illegible]

China-pub.com [illegible]

60. [illegible] 14[illegible]

61. [illegible]

62. [illegible]“[illegible]”[illegible]

[illegible] C [illegible]

[illegible]

[illegible] “ [illegible] ” [illegible] “ [illegible] ” [illegible]

C.1 [illegible]

[illegible] “ shapes” [illegible] draw() [illegible] draw() [illegible] 14 [illegible]

[illegible] “ [illegible] ” [illegible] VTABLE [illegible] VPTR [illegible]

[illegible]

[illegible] shape [illegible] circle [illegible] square [illegible] triangle [illegible]

Coplien [1] [illegible] “ [illegible] ” [illegible] “ [illegible] ” [illegible]

[illegible] shape [illegible]

```
//: SSHAPE.CPP -- "Virtual constructors"
// Used in a simple "shape" framework
#include <iostream.h>
#include "..\14\tstash.h"

class shape {
```

---

[1] James O.Coplien [illegible] C++ [illegible] Addison_Wesley, 1992 [illegible]

```
  shape* S;
  // Prevent copy-construction & operator=
  shape(shape&);
  shape operator=(shape&);
protected:
  shape() { S = 0; };
public:
  enum type { Circle, Square, Triangle };
  shape(type); // "Virtual" constructor
  virtual void draw() { S->draw(); }
  virtual ~shape() {
    cout << "~shape\n";
    delete S;
  }
};
class circle : public shape {
  // Prevent copy-construction & operator=
  circle(circle&);
  circle operator=(circle&);
public:
  circle() {}
  void draw() { cout << "circle::draw\n"; }
  ~circle() { cout << "~circle\n"; }
};

class square : public shape {
  // Prevent copy-construction & operator=
  square(square&);
  square operator=(square&);
public:
  square() {}
  void draw() { cout << "square::draw\n"; }
  ~square() { cout << "~square\n"; }
};

class triangle : public shape {
  // Prevent copy-construction & operator=
  triangle(triangle&);
  triangle operator=(triangle&);
public:
  triangle() {}
  void draw() { cout << "triangle::draw\n"; }
  ~triangle() { cout << "~triangle\n"; }
};

shape::shape(type t) {
  switch(t) {
    case Circle: s = new circle; break;
    case Square: s = new square; break;
    case Triangle: s = new triangle; break;
  }
  draw();  // Virtual call in the constructor
}

main() {
  tstash<shape> shapes; // Default to ownership
  cout << "virtual constructor calls:" << endl;
  shapes.add(new shape(shape::Circle));
  shapes.add(new shape(shape::Square));
  shapes.add(new shape(shape::Triangle));
  cout << "virtual function calls:" << endl;
  for(int i = 0; i < shapes.count(); i++)
    shapes[i]->draw();
  shape c(shape::Circle); // Can create on stack
}
```

[illegible] shape[illegible] shape[illegible]“[illegible]”[illegible]

type[illegible] shape[illegible] shape[illegible] shape[illegible] type [illegible] shape[illegible] shape[illegible] shape[illegible] 3[illegible]“ cheshire cat”[illegible]

[illegible]——[illegible]“[illegible]”[illegible]

“[illegible]” shape[illegible] type[illegible] class shape[illegible] protected[illegible] shape[illegible] shape(type)[illegible] shape[illegible]

[illegible]——[illegible] S[illegible] 0[illegible]——[illegible] Coplien[illegible]“[illegible]”[illegible]“[illegible]”——[illegible]“[illegible]”[illegible] S[illegible]“[illegible]”[illegible] S[illegible]“[illegible]”[illegible] S[illegible]“[illegible]”[illegible] S[illegible] 0[illegible]

“[illegible]”[illegible]

[illegible] switch[illegible]“[illegible]”[illegible]“[illegible]”[illegible]

[illegible] draw()[illegible] shape[illegible] draw()[illegible]“[illegible]”[illegible] S[illegible]“[illegible]”[illegible] draw()[illegible] shape[illegible]“[illegible]”[illegible]“[illegible]”[illegible]

[illegible] main()[illegible] s[][illegible]“[illegible]” shape[illegible] VPTR[illegible] VPTR[illegible] shape[illegible] circle[illegible] square [illegible] triangle[illegible]

[illegible] for[illegible] shape[illegible] draw()[illegible] VPTR[illegible] shape[illegible] draw[illegible] draw()[illegible]“[illegible]”[illegible] S[illegible]“[illegible]”[illegible] draw()[illegible] shape[illegible]“[illegible]”[illegible]

[illegible]

[illegible]“[illegible]”[illegible] this[illegible]“[illegible]”[illegible]“[illegible]”[illegible]——[illegible]“[illegible]”——[illegible] S[illegible]“[illegible]”[illegible] S[illegible] 0[illegible] protected[illegible] S[illegible] 0[illegible] 0[illegible]“[illegible]”[illegible]

1) [illegible]

2) [illegible]

[illegible]“[illegible]”[illegible]“[illegible]”[illegible] this[illegible]“[illegible]”[illegible] this[illegible]“[illegible]”[illegible] S[illegible] 0[illegible]

[illegible] C++[illegible]

- [illegible]

[illegible] shape[illegible] delete[illegible]——[illegible] square[illegible]——[illegible]

□□□□□□□□ delete□□□□□□□□□□□□□ main()□□ delete s [i]□□□□□
□□S[i]□□□□□□□shape□□□□□□□□□□□□□□□□□□□shape□□□□□□□□□□□□□□□□□□□□□□□□□square□□□□□□□□□□□□□□□□□“□□□□□”□□□□□□□□□□□□□□□□□□shape□□□□□□□□□□□□□□“□□”□□□□□□□□□□□□□□□□□□□□□□□shape□□□□VPTR□shape□VPTR□□□□square□□□□□□□□□□□□□shape□□□□□□□□□□“□□”□□□S□□delete□□S□□□□□□square□□□□□□□□□□□□□□□□□□□□□□square□□□□□□

□□□□□□□□□□□□□□□□□□□□□square□□delete S□□□□□□□□□□□□□□□□□□□□□□□□“□□”S□□□□□□□□□“□□”S□□□□□“□□”□□“□□”□□□□□□□□□□□□□□□□□□□□□□□□□□□□delete□□□□□□□□

□□□□□□□□□□□□□□“□□”□S□□□□□□□□□□□“□□”□□□□□□□□□□□□□□□delete 0□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□protected□□□□□“□□”□□□□□□□□□□□□□□S□□□□□□□□□□

C.2 □□□□□□

“□□□□□”□□“□□——□□”□□□

```
//: SSHAPE2.CPP -- Alternative to SSHAPE.CPP
#include <iostream.h>
#include "..\14\tstash.h"

class shape {
  shape(shape&); // Prevent copy-construction
protected:
  shape() {} // Prevent stack objects
  // But allow access to derived constructors
public:
  enum type { Circle, Square, Triangle };
  virtual void draw() = 0;
  virtual ~shape() { cout << "~shape\n"; }
  static shape* make(type);
};

class circle : public shape {
  circle(circle&); // No copy-construction
  circle operator=(circle&); // No operator=
protected:
  circle() {};
```

```
public:
  void draw() { cout << "circle::draw\n"; }
  ~circle() { cout << "~circle\n"; }
  friend shape* shape::make(type t);
};

class square : public shape {
  square(square&); // No copy-construction
  square operator=(square&); // No operator=
protected:
  square() {};
public:
  void draw() { cout << "square::draw\n"; }
  ~square() { cout << "~square\n"; }
  friend shape* shape::make(type t);
};

class triangle : public shape {
  triangle(triangle&); // No copy-construction
  triangle operator=(triangle&); // Prevent
protected:
  triangle() {};
public:
  void draw() { cout << "triangle::draw\n"; }
  ~triangle() { cout << "~triangle\n"; }
  friend shape* shape::make(type t);
};

shape* shape::make(type t) {
  shape* S;
  switch(t) {
    case Circle: S = new circle; break;
    case Square: S = new square; break;
    case Triangle: S = new triangle; break;
  }
  S->draw(); // Virtual function call
  return S;
}

main() {
  tstash<shape> shapes; // Default to ownership
  shapes.add(shape::make(shape::Circle));
  shapes.add(shape::make(shape::Square));
  shapes.add(shape::make(shape::Triangle));
  cout << "virtual function calls:\n";
  for(int i = 0; i < shapes.count(); i++)
  shapes[i]->draw();
//Circle c; // error: can't create on stack
}
```

□□□enum□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□protected□□□protected□□□□□□□□□□□□□□□□

Educate Different

Powered by XiaoGuo’s publishing Studio

QQ:8204136

Website: www.mcuzone.com

2005